TRICKS & TIPS

C++ & Python

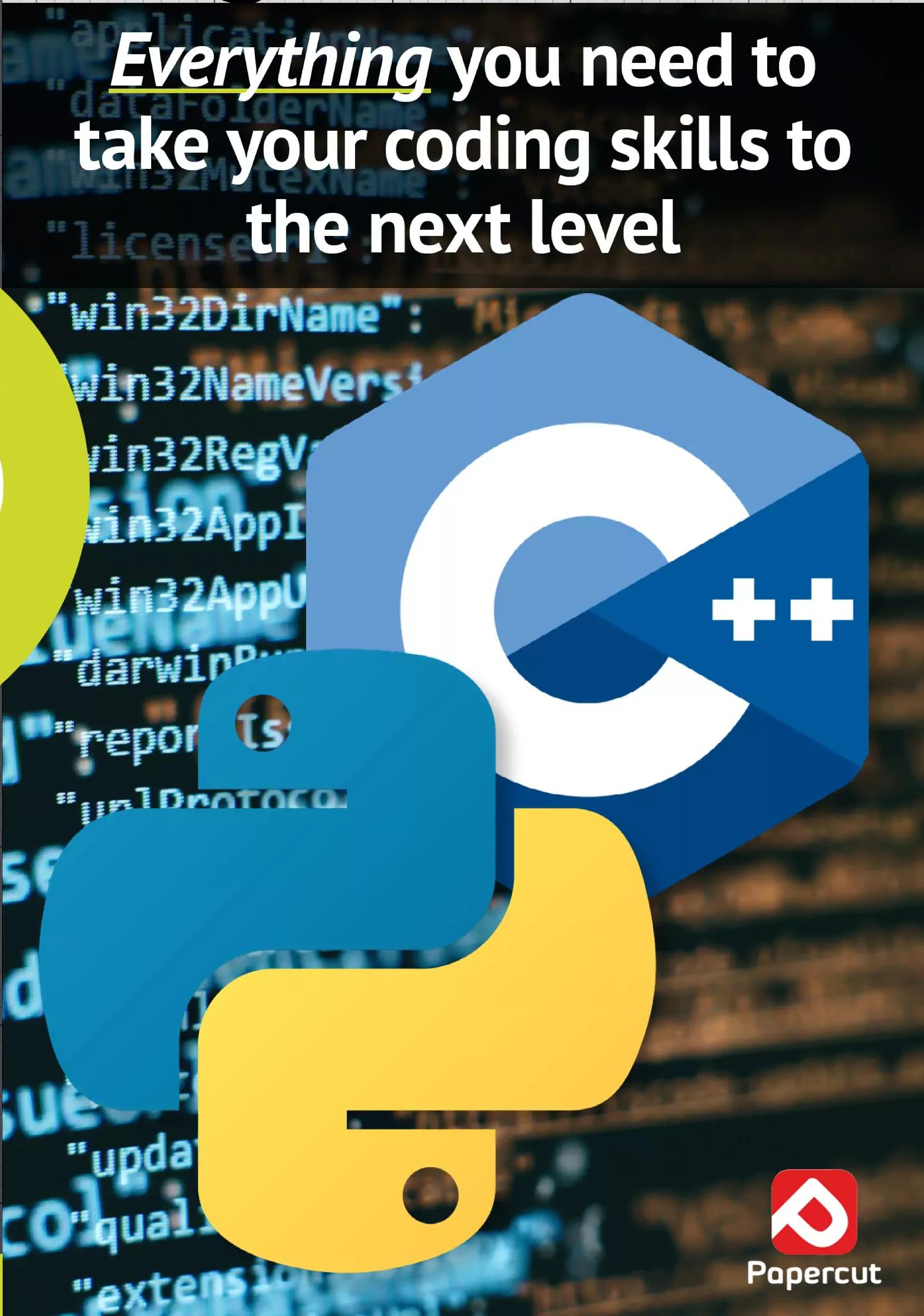
- Next level secrets and fixes get to the heart of the C++ coding language

OVER 445 SECRETS & HACKS

- We share our awesome tips and shortcuts for coding Python and C++
- Advanced guides and tutorials for programming with C++ & Python

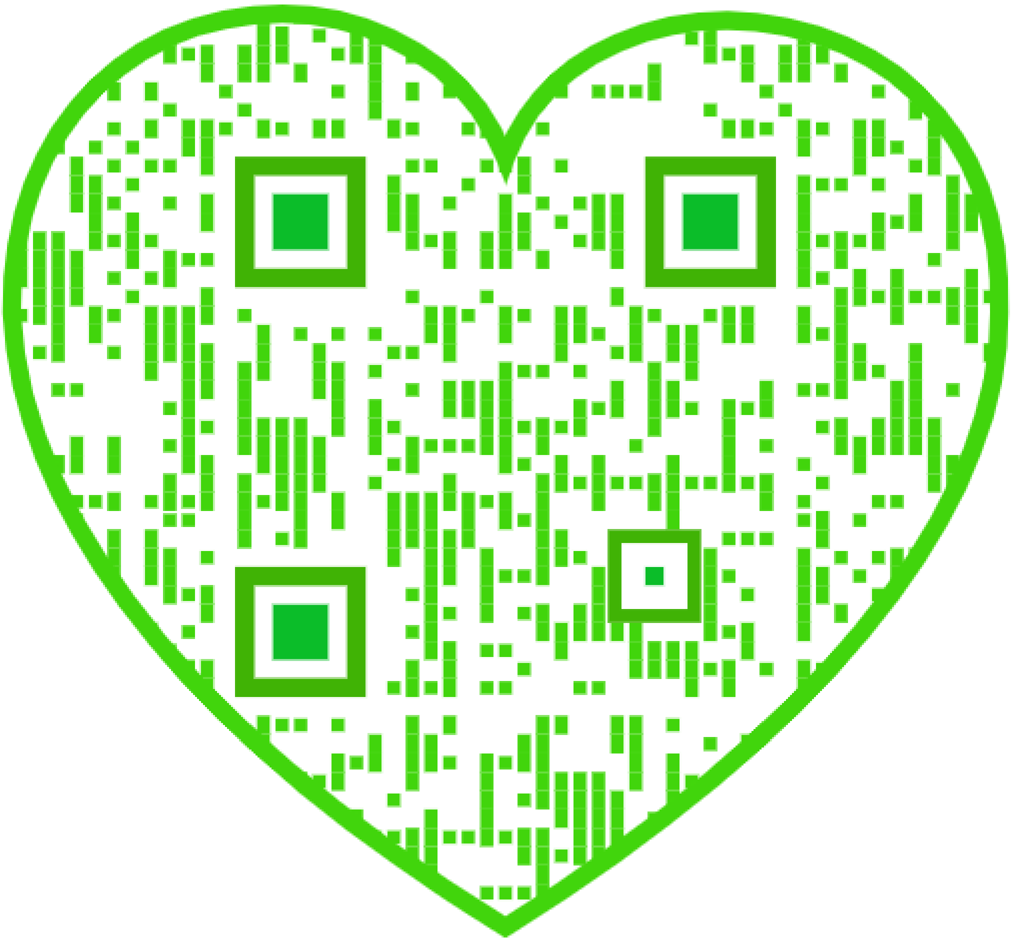
100% INDEPENDENT

Everything you need to take your coding skills to the next level



```
"win32DirName":
"Win32NameVers
"win32RegV
"win32AppI
"win32AppU
"darwin
"repor
"upda
"qual
"extensi
```

Papercut



Save a whopping

25% Off!

ALL Tech Manuals

with Papercut

Not only can you learn new skills and master your tech, but you can now SAVE 25% off all of our coding and consumer tech digital and print guidebooks!

Simply use the following exclusive code at checkout:

NYHF23CN

TRICKS
& TIPS

C++ & Python

C++ & Python Tricks & Tips is the perfect digital publication for the user that wants to take their skill set to the next level. Do you want to enhance your user experience? Or wish to gain insider knowledge? Do you want to learn directly from experts in their field? Learn the numerous short cuts that the professionals use? Over the pages of the new advanced user guide you will learn everything you will need to know to become a more confident, better skilled and experienced owner. A user that will make the absolute most of their coding use and ultimately the platforms themselves. An achievement you can earn by simply enabling us to exclusively help and teach you the abilities we have gained over our decades of experience.

Over the page our journey continues, and we will be with you at every stage to advise, inform and ultimately inspire you to go further.

Contents

6 Working with Data

8 Lists
10 Tuples
12 Dictionaries
14 Splitting and Joining Strings
16 Formatting Strings
18 Date and Time
20 Opening Files
22 Writing to Files
24 Exceptions
26 Python Graphics

28 Using Modules

30 Calendar Module
32 OS Module
34 Random Module
36 Tkinter Module
38 Pygame Module
42 Create Your Own Modules

44 C++ Input/Output

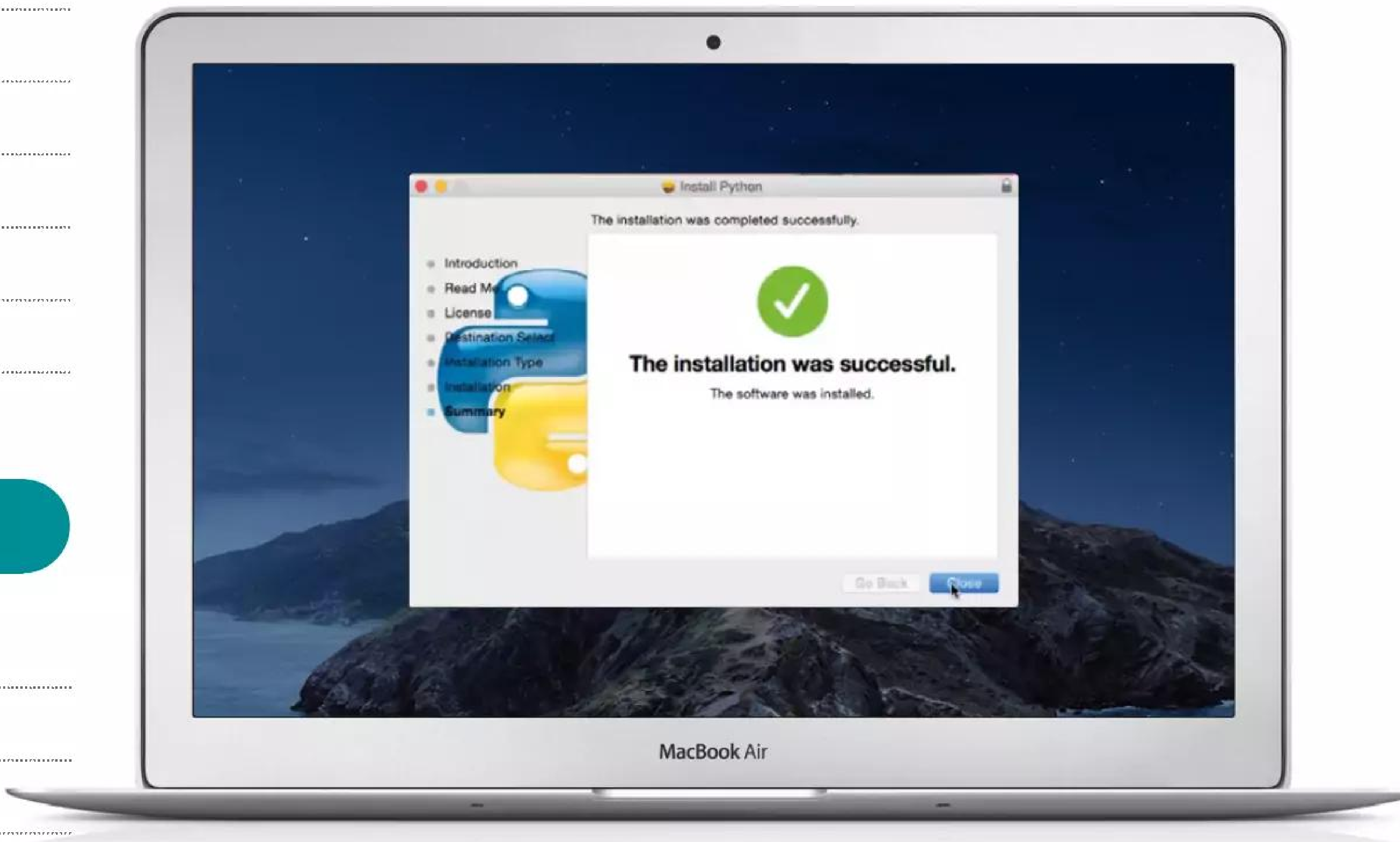
46 User Interaction
48 Character Literals
50 Defining Constants
52 File Input/Output

54 Loops and Decision Making

56 While Loop
58 For Loop
60 Do… While Loop
62 If Statement
64 If… Else Statement

66 Working with Code

68 Common Coding Mistakes
70 Beginner Python Mistakes
72 Beginner C++ Mistakes
74 Where Next?


Install Python
The installation was completed successfully.
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
The installation was successful.
The software was installed.
Go Back
Close
MacBook Air

> "...We've taken two of the most powerful and versatile programming languages available, Python and C++ and broken them down into bite-sized tutorials and guides to help you learn how they work, and how to make them work for you..."

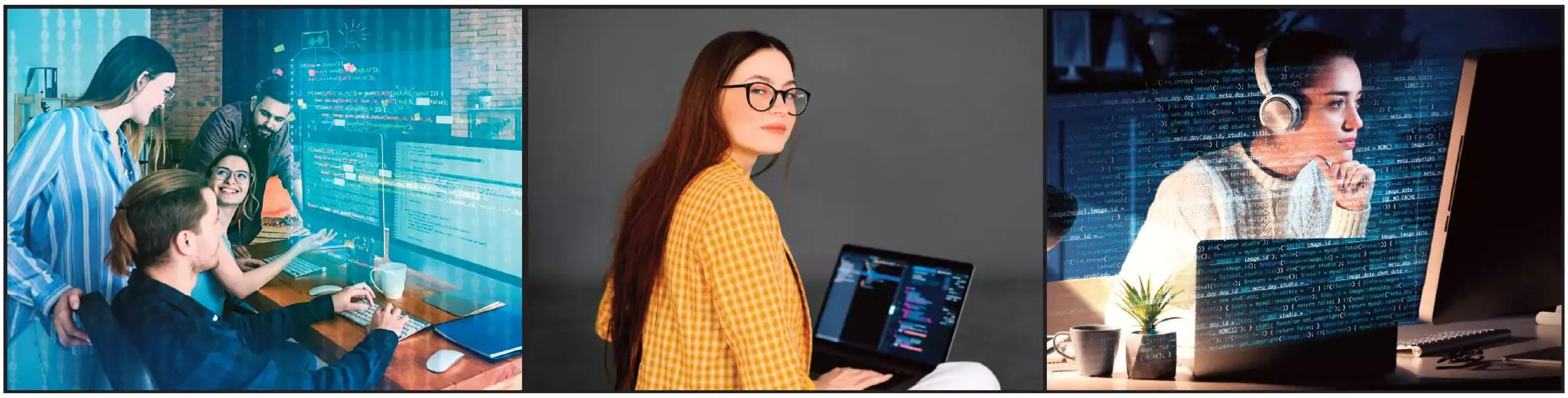
FREE CODE DOWNLOAD!

50 Complete Programs

Over 20,000 lines of code!

Visit: www.pclpublications.com/exclusives

Please note: Sign up is required to download



Working with Data

Data is everything. With it you can display, control, add, remove, create and manipulate Python to your every demand. Over these coming pages we look at how you can create lists, tuples, dictionaries and multi-dimensional lists; and see how to use them to forge exciting and useful programs.

Then, you can learn how to use date and time functions, write to files in your system and even create graphical user interfaces that take your coding skills to new levels and into new project ideas.

Lists

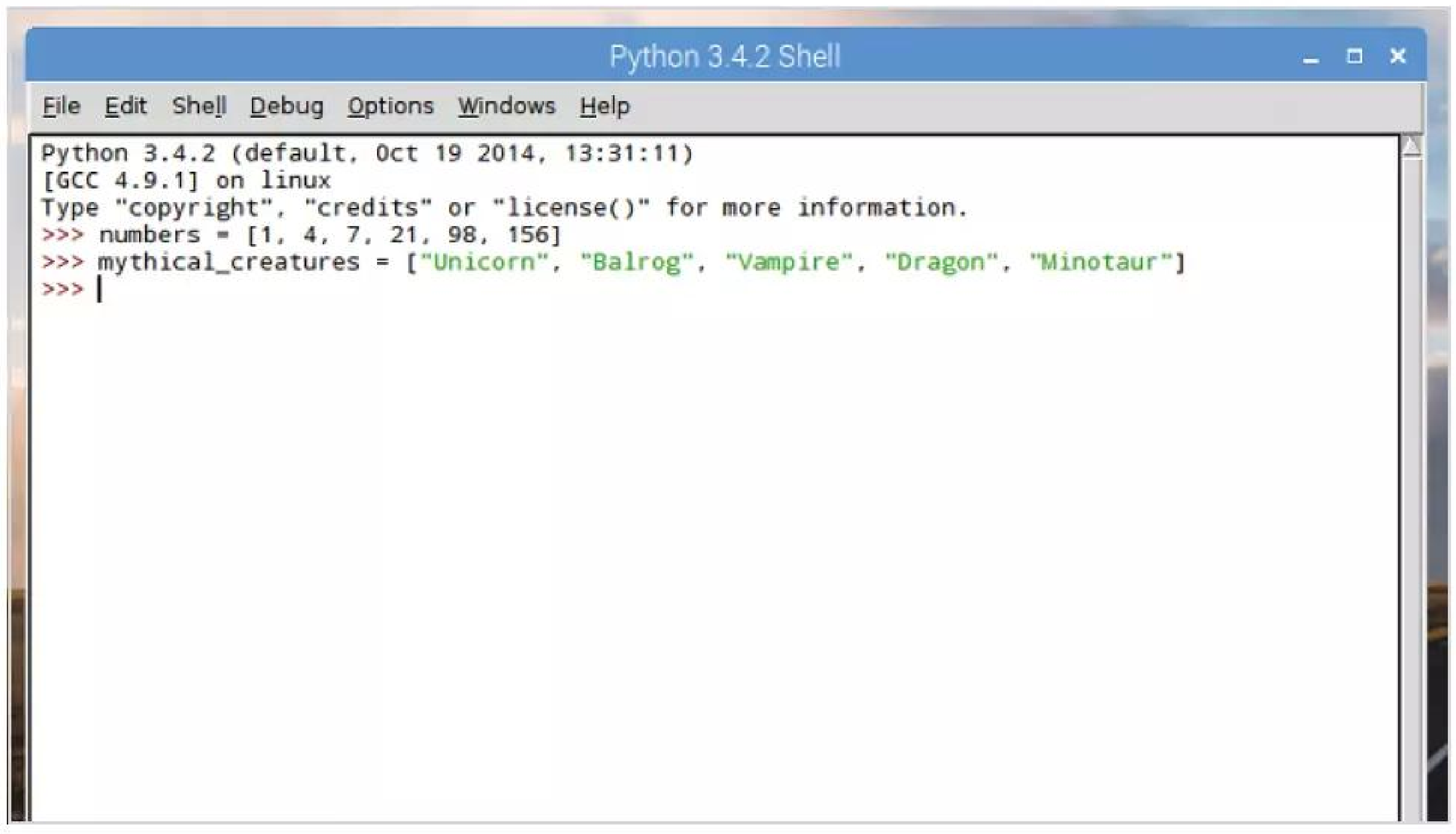
Lists are one of the most common types of data structures you will comes across in Python. A list is simply a collection of items, or data if you prefer, that can be accessed as a whole, or individually if wanted.

WORKING WITH LISTS

Lists are extremely handy in Python. A list can be strings, integers and also variables. You can even include functions in lists, and lists within lists.

STEP 1 A list is a sequence of data values called items. You create the name of your list followed by an equals sign, then square brackets and the items separated by commas; note that strings use quotes:

```
numbers = [1, 4, 7, 21, 98, 156]
mythical_creatures – [“Unicorn”, “Balrog”,
“Vampire”, “Dragon”, “Minotaur”]
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> |
```

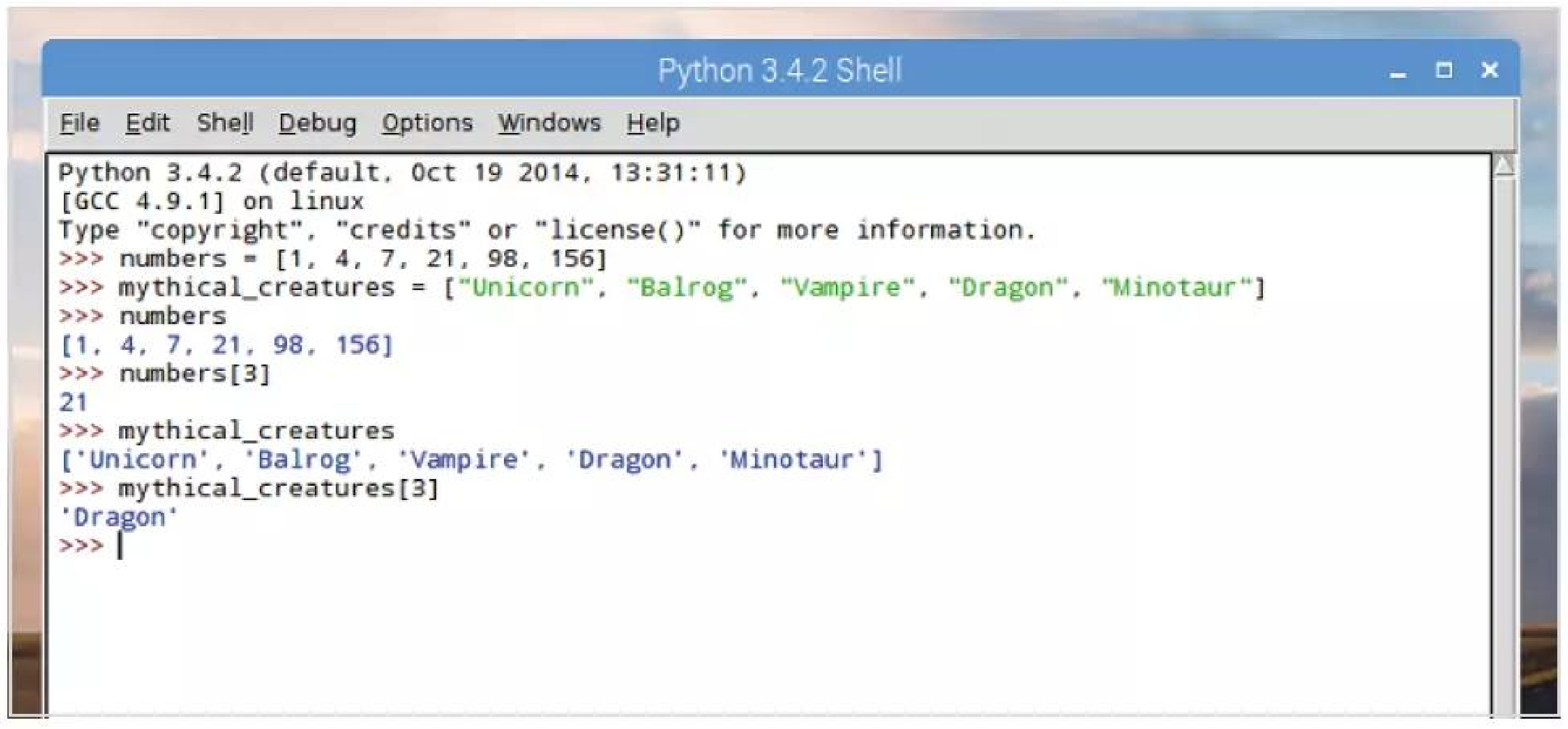
STEP 2 Once you've defined your list you can call each by referencing its name, followed by a number. Lists start the first item entry as 0, followed by 1, 2, 3 and so on. For example:

`numbers`

To call up the entire contents of the list.

`numbers[3]`

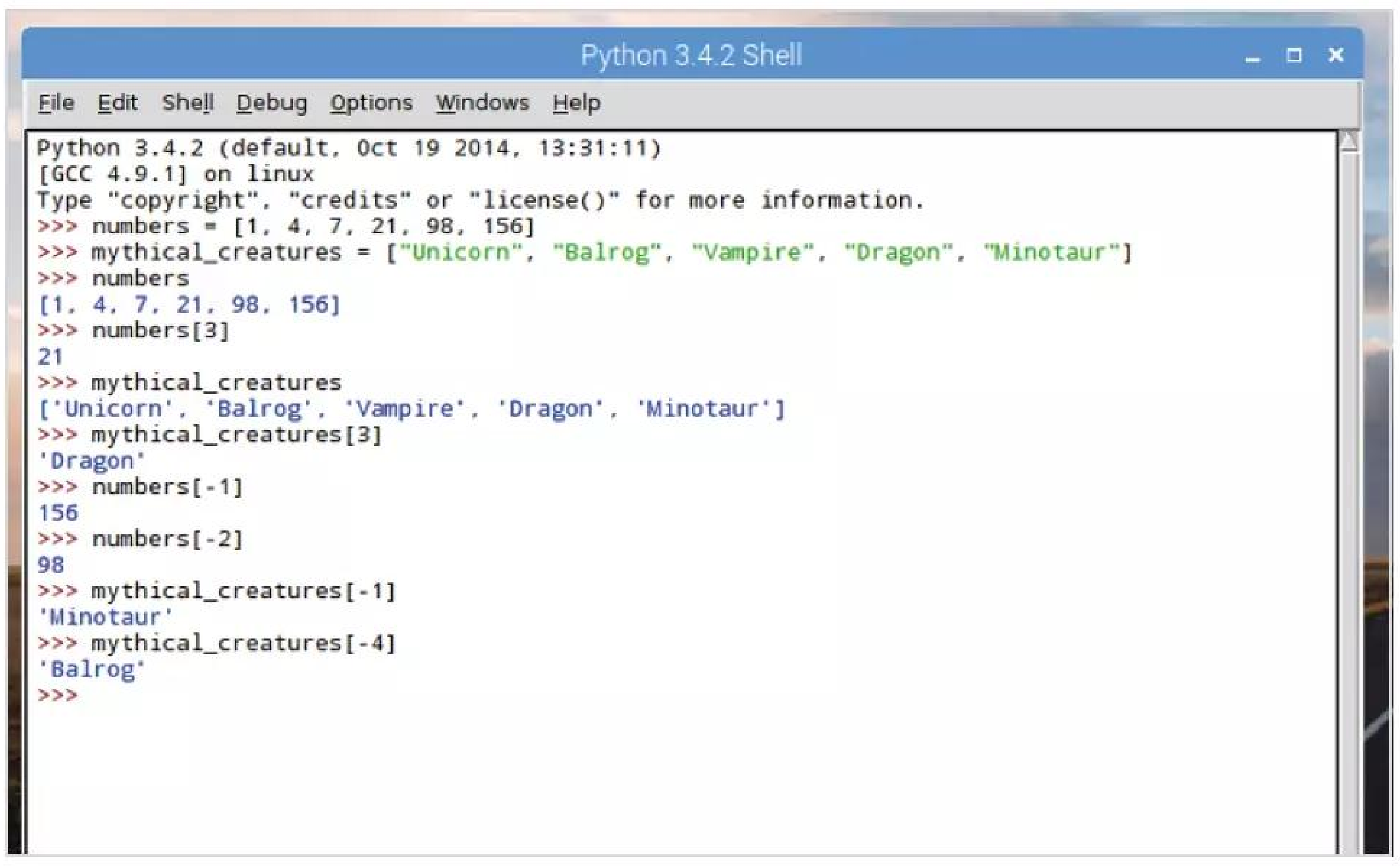
To call the third from zero item in the list (21 in this case).



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers[3]
21
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> mythical_creatures[3]
'Dragon'
>>> |
```

STEP 3 You can also access, or index, the last item in a list by using the minus sign before the item number [-1], or the second to last item with [-2] and so on. Trying to reference an item that isn't in the list, such as [10] will return an error:

```
numbers[-1]
mythical_creatures[-4]
```

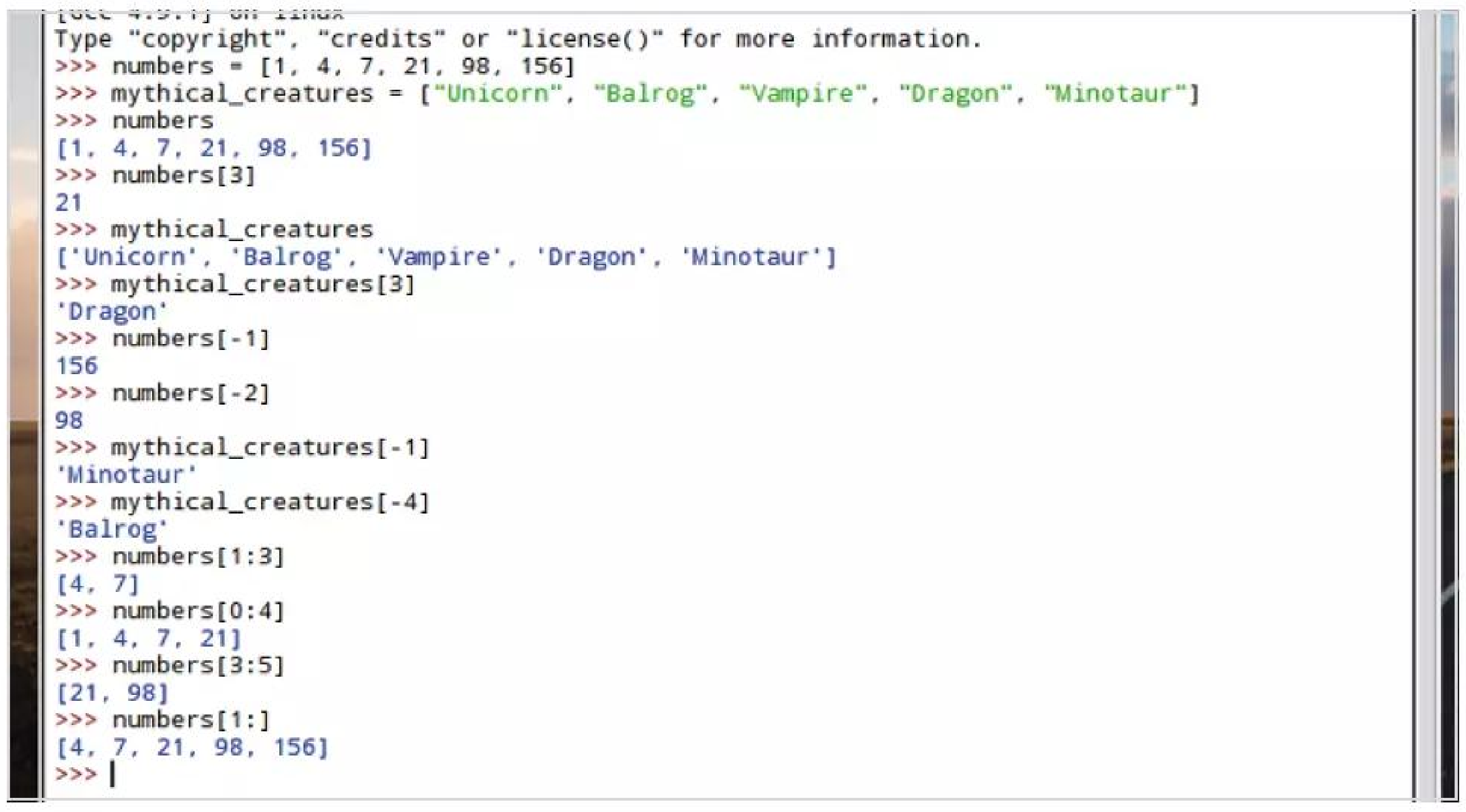


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers[3]
21
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> mythical_creatures[3]
'Dragon'
>>> numbers[-1]
156
>>> numbers[-2]
98
>>> mythical_creatures[-1]
'Minotaur'
>>> mythical_creatures[-4]
'Balrog'
>>>
```

STEP 4 Slicing is similar to indexing but you can retrieve multiple items in a list by separating item numbers with a colon. For example:

`numbers[1:3]`

Will output the 4 and 7, being item numbers 1 and 2. Note that the returned values don't include the second index position (as you would numbers[1:3] to return 4, 7 and 21).



```
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers[3]
21
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> mythical_creatures[3]
'Dragon'
>>> numbers[-1]
156
>>> numbers[-2]
98
>>> mythical_creatures[-1]
'Minotaur'
>>> mythical_creatures[-4]
'Balrog'
>>> numbers[1:3]
[4, 7]
>>> numbers[0:4]
[1, 4, 7, 21]
>>> numbers[3:5]
[21, 98]
>>> numbers[1:]
[4, 7, 21, 98, 156]
>>> |
```

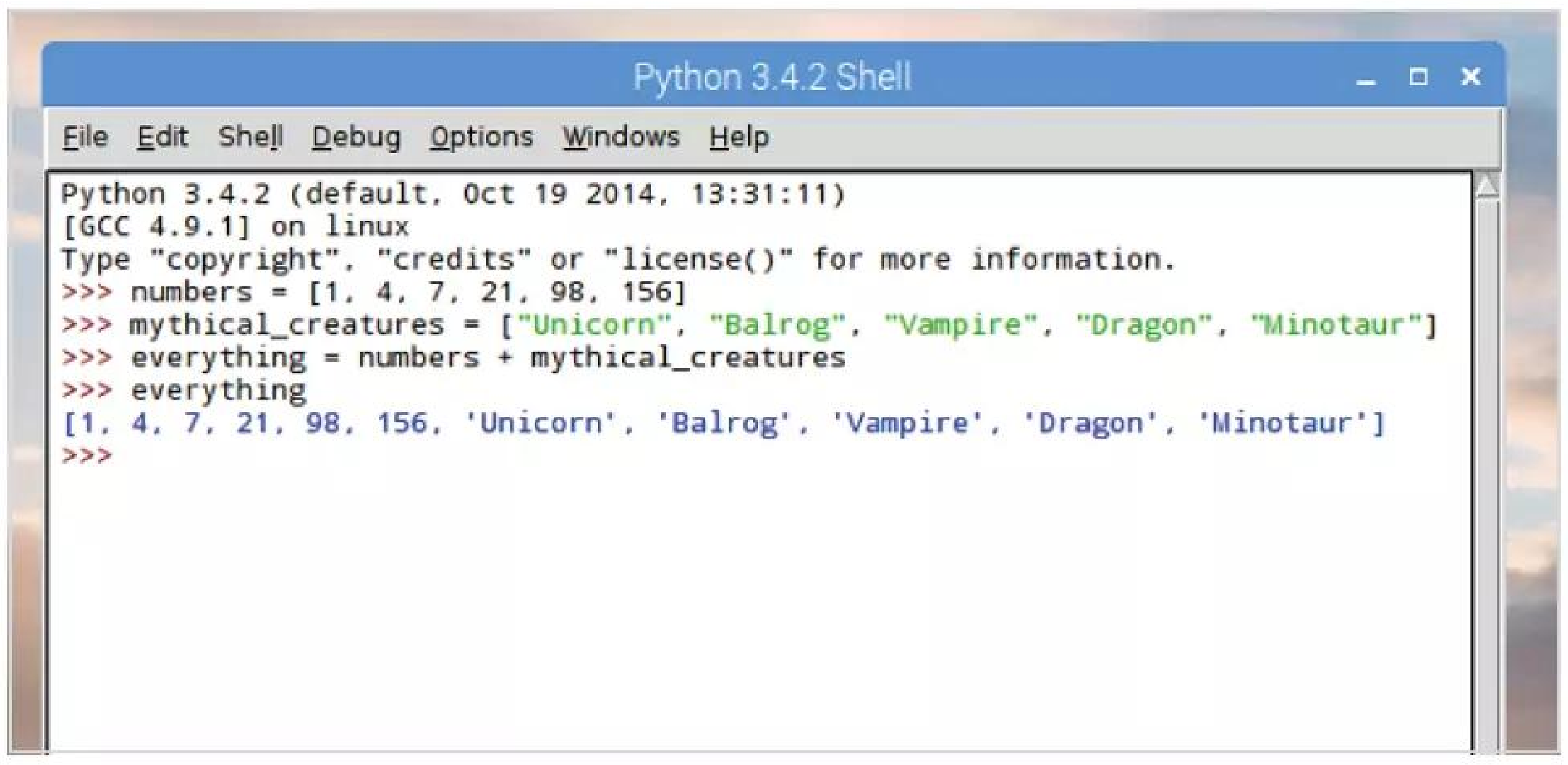
STEP 5

You can update items within an existing list, remove items and even join lists together. For example, to join two lists you can use:

```
everything = numbers + mythical_creatures
```

Then view the combined list with:

```
everything
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> everything = numbers + mythical_creatures
>>> everything
[1, 4, 7, 21, 98, 156, 'Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>>
```

STEP 6

Items can be added to a list by entering:

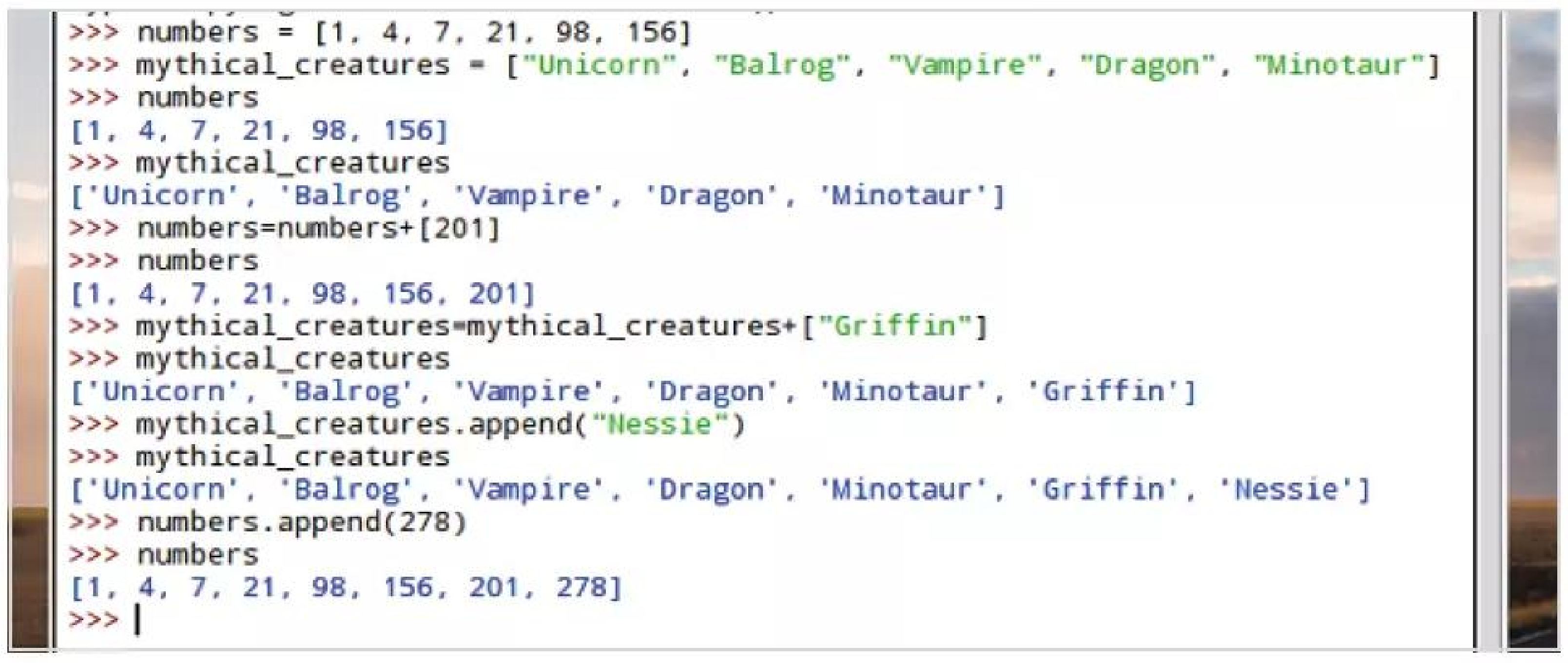
```
numbers=numbers+[201]
```

Or for strings:

```
mythical_creatres=mythical_creatures+[“Griffin”]
```

Or by using the append function:

```
mythical_creatures.append(“Nessie”)
numbers.append(278)
```



```
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> numbers=numbers+[201]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures=mythical_creatures+["Griffin"]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> mythical_creatures.append("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin', 'Nessie']
>>> numbers.append(278)
>>> numbers
[1, 4, 7, 21, 98, 156, 201, 278]
>>> |
```

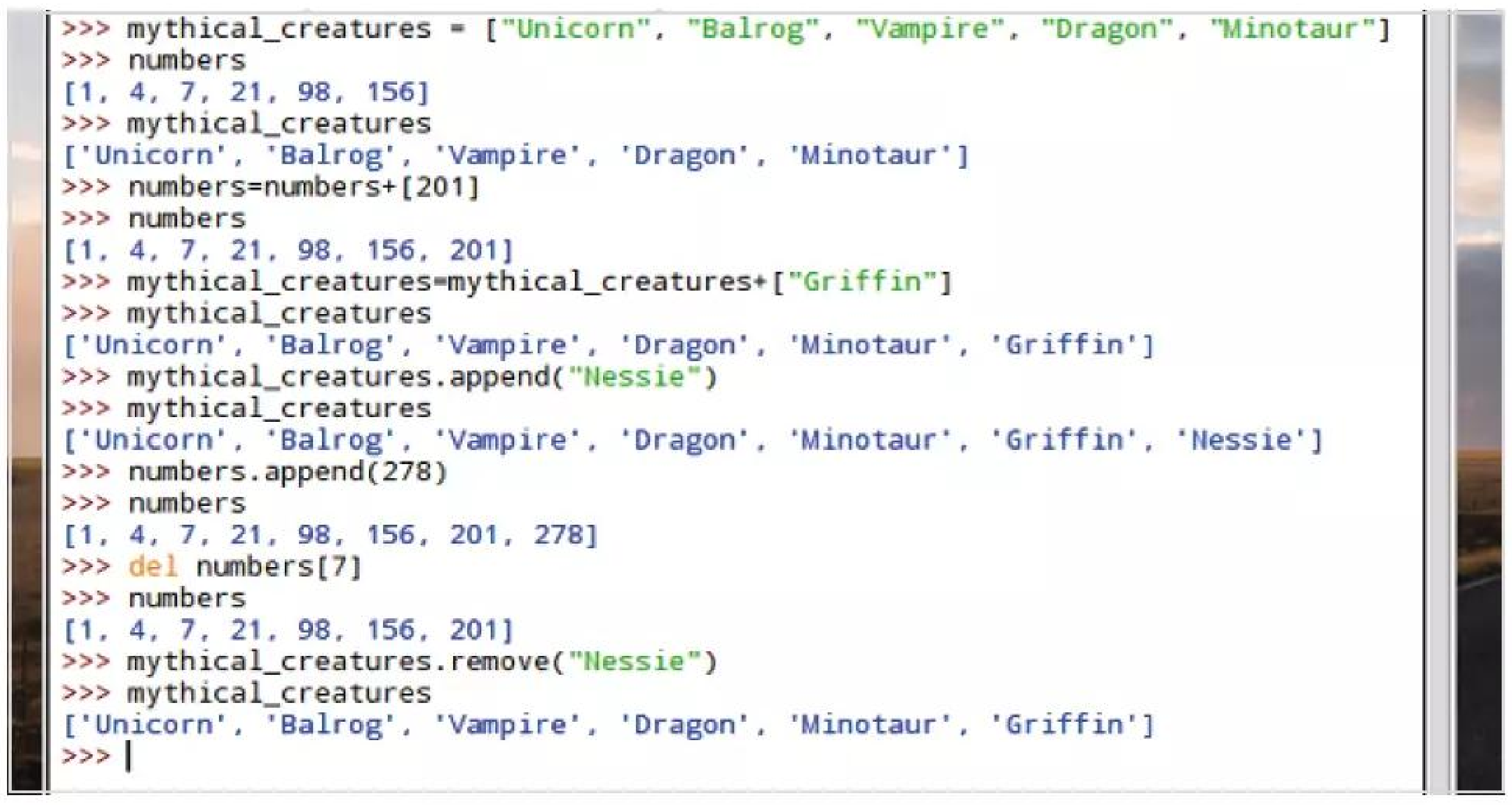
STEP 7

Removal of items can be done in two ways. The first is by the item number:

```
del numbers[7]
```

Alternatively, by item name:

```
mythical_creatures.remove(“Nessie”)
```



```
>>> mythical_creatures = ["Unicorn", "Balrog", "Vampire", "Dragon", "Minotaur"]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur']
>>> numbers=numbers+[201]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures=mythical_creatures+["Griffin"]
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> mythical_creatures.append("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin', 'Nessie']
>>> numbers.append(278)
>>> numbers
[1, 4, 7, 21, 98, 156, 201, 278]
>>> del numbers[7]
>>> numbers
[1, 4, 7, 21, 98, 156, 201]
>>> mythical_creatures.remove("Nessie")
>>> mythical_creatures
['Unicorn', 'Balrog', 'Vampire', 'Dragon', 'Minotaur', 'Griffin']
>>> |
```

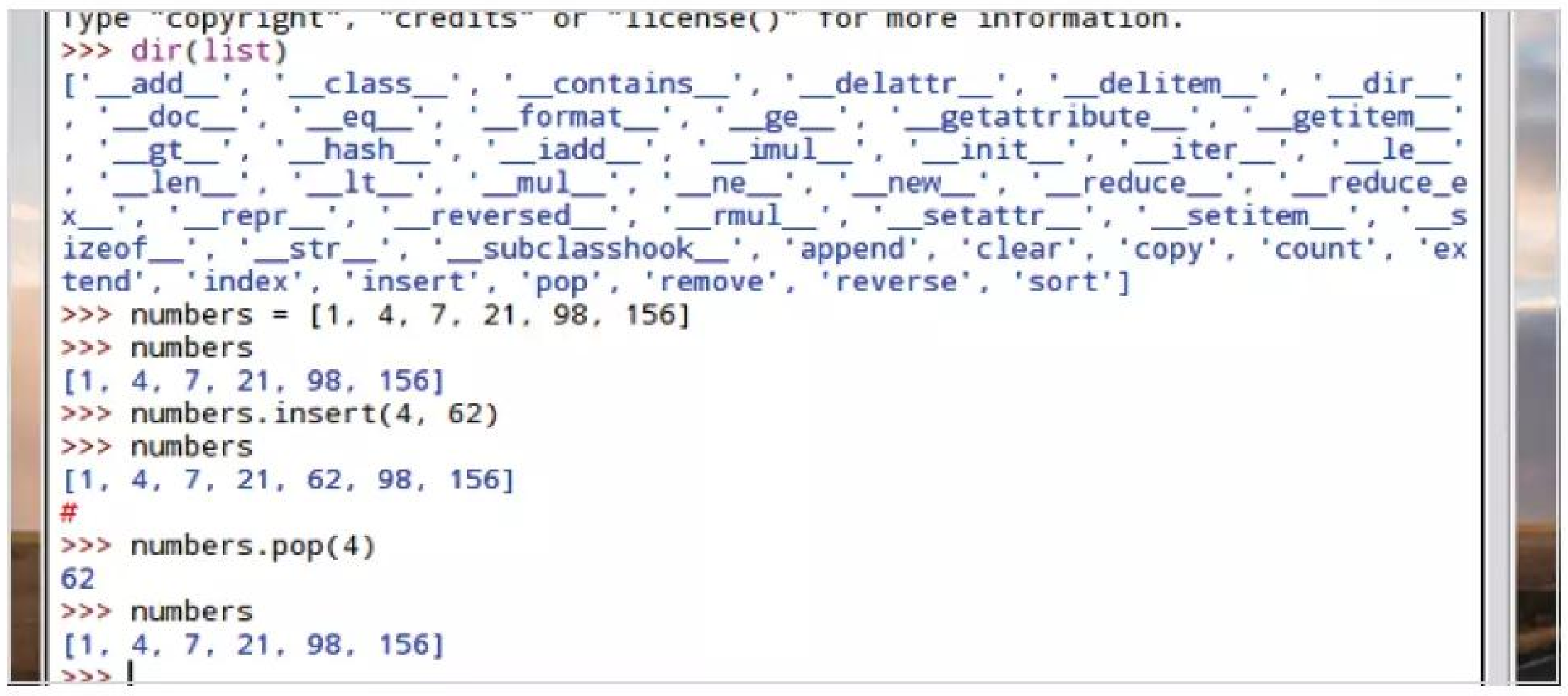
STEP 8

You can view what can be done with lists by entering dir(list) into the Shell. The output is the available functions, for example, insert and pop are used to add and remove items at certain positions. To insert the number 62 at item index 4:

```
numbers.insert(4, 62)
```

To remove it:

```
numbers.pop(4)
```



```
Type "copyright", "credits" or "license()" for more information.
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__'
, '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__'
, '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__'
, '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__s
izeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'ex
tend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> numbers = [1, 4, 7, 21, 98, 156]
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> numbers.insert(4, 62)
>>> numbers
[1, 4, 7, 21, 62, 98, 156]
#
>>> numbers.pop(4)
62
>>> numbers
[1, 4, 7, 21, 98, 156]
>>> |
```

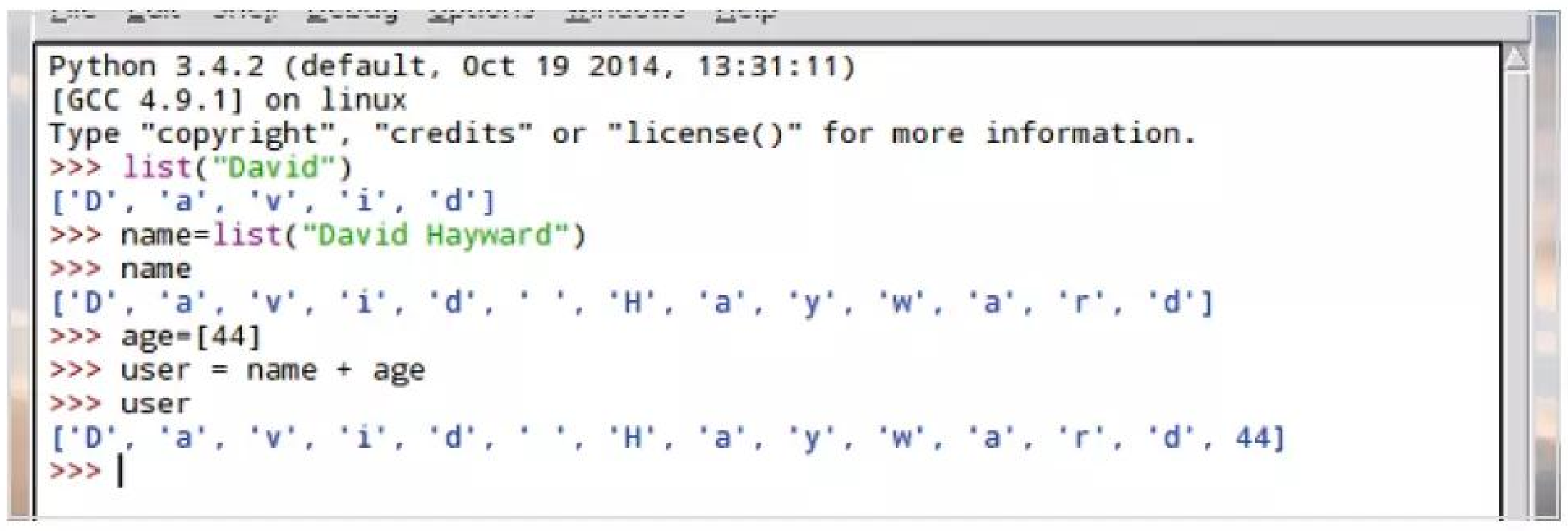
STEP 9

You also use the list function to break a string down into its components. For example:

```
list(“David”)
```

Breaks the name David into ‘D’, ‘a’, ‘v’, ‘i’, ‘d’. This can then be passed to a new list:

```
name=list(“David Hayward”)
name
age=[44]
user = name + age
user
```



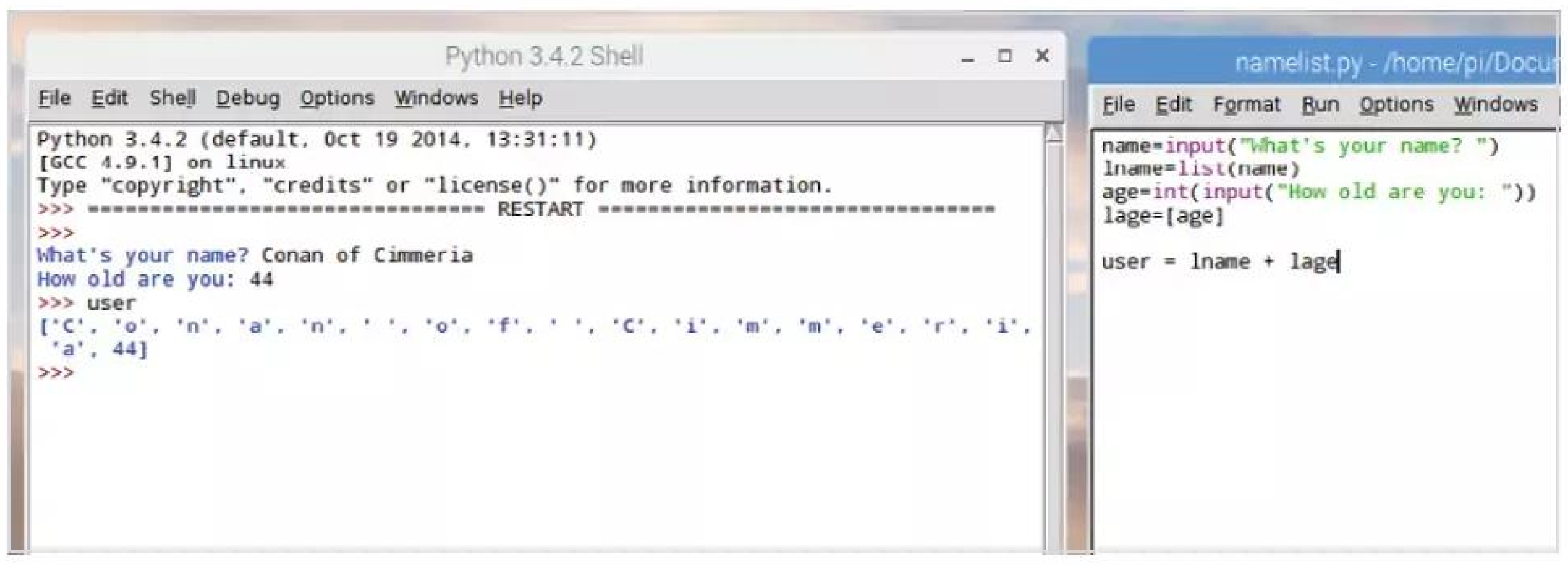
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list("David")
['D', 'a', 'v', 'i', 'd']
>>> name=list("David Hayward")
>>> name
['D', 'a', 'v', 'i', 'd', ' ', 'H', 'a', 'y', 'w', 'a', 'r', 'd']
>>> age=[44]
>>> user = name + age
>>> user
['D', 'a', 'v', 'i', 'd', ' ', 'H', 'a', 'y', 'w', 'a', 'r', 'd', 44]
>>> |
```

STEP 10

Based on that, you can create a program to store someone’s name and age as a list:

```
name=input(“What’s your name? “)
lname=list(name)
age=int(input(“How old are you: “))
lage=[age]

user = lname + lage
```

The combined name and age list is called user, which can be called by entering user into the Shell. Experiment and see what you can do.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
namelist.py - /home/pi/Docu
File Edit Format Run Options Windows

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What's your name? Conan of Cimmeria
How old are you: 44
>>> user
['C', 'o', 'n', 'a', 'n', ' ', 'o', 'f', ' ', 'C', 'i', 'm', 'm', 'e', 'r', 'i',
 'a', 44]
>>>
```

```
name=input("What's your name? ")
lname=list(name)
age=int(input("How old are you: "))
lage=[age]

user = lname + lage
```

Tuples

Tuples are very much identical to lists. However, where lists can be updated, deleted or changed in some way, a tuple remains a constant. This is called immutable and they're perfect for storing fixed data items.

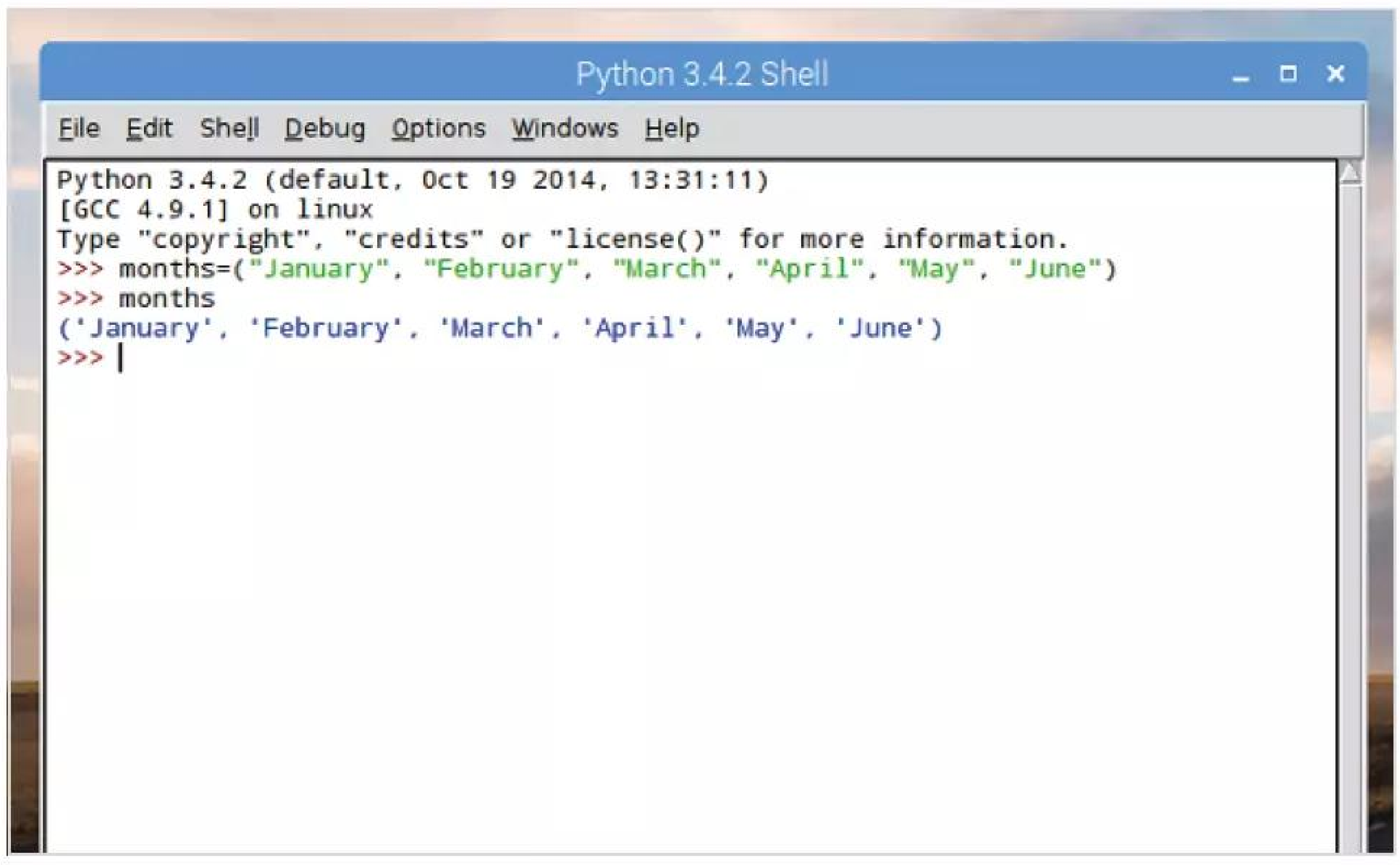
THE IMMUTABLE TUPLE

Reasons for having tuples vary depending on what the program is intended to do. Normally, a tuple is reserved for something special but they're also used for example, in an adventure game, where non-playing character names are stored.

STEP 1

A tuple is created the same way as a list but in this instance you use curved brackets instead of square brackets. For example:

```
months=("January", "February", "March", "April",
"May", "June")
months
```



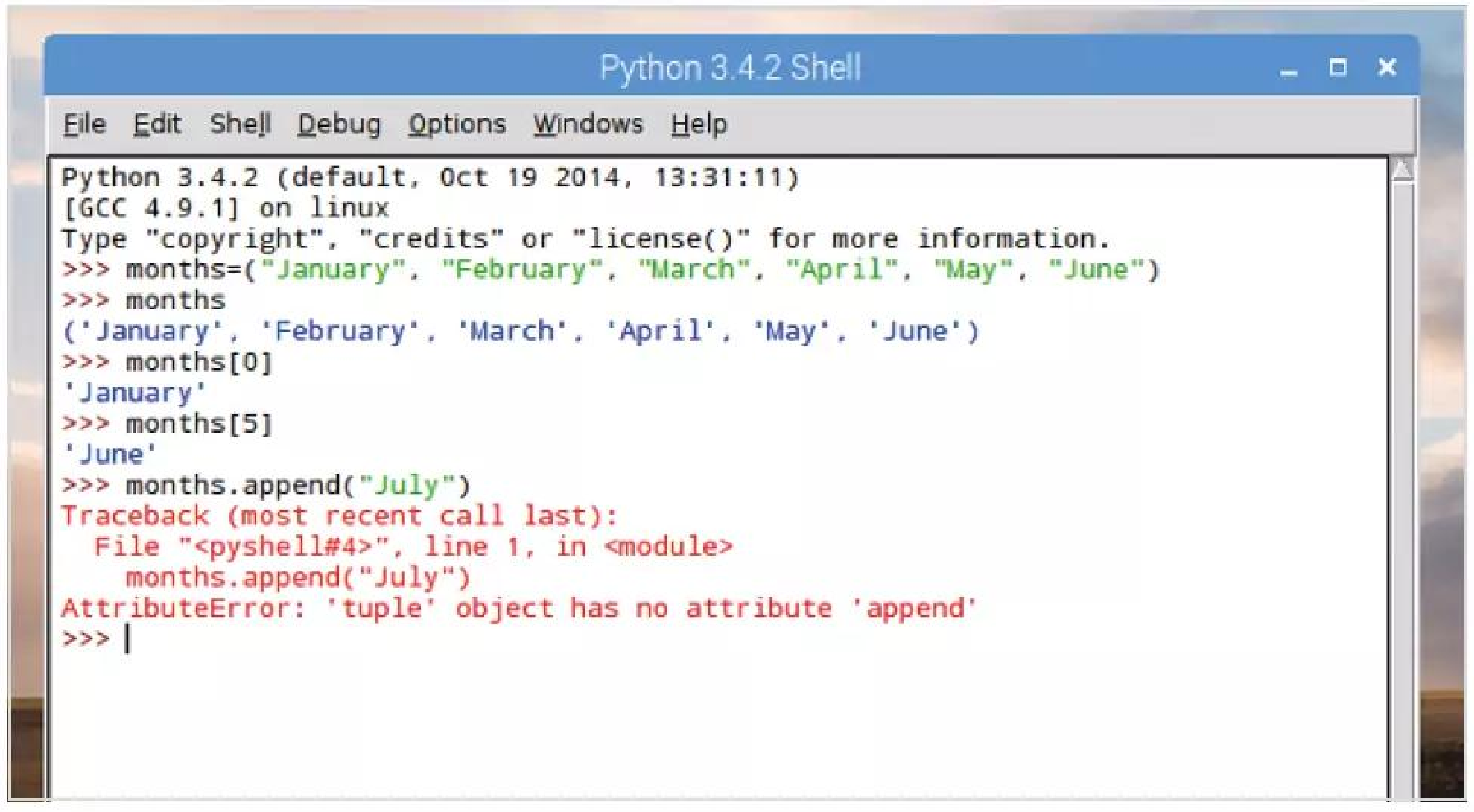
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> months=("January", "February", "March", "April", "May", "June")
>>> months
('January', 'February', 'March', 'April', 'May', 'June')
>>> 
```

STEP 2

Just as with lists, the items within a named tuple can be indexed according to their position in the data range, i.e.:

```
months[0]
months[5]
```

However, any attempt at deleting or adding to the tuple will result in an error in the Shell.

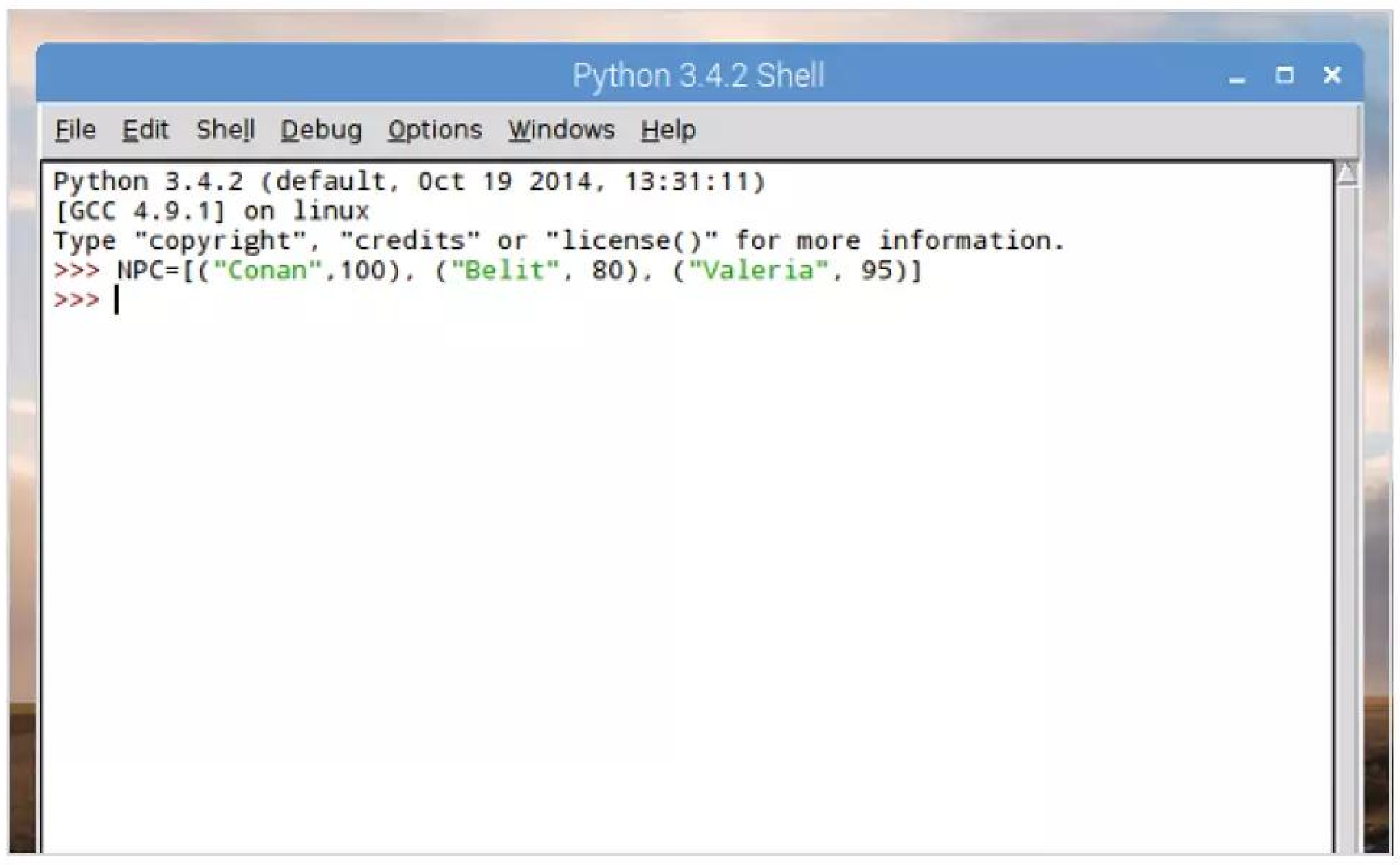


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> months=("January", "February", "March", "April", "May", "June")
>>> months
('January', 'February', 'March', 'April', 'May', 'June')
>>> months[0]
'January'
>>> months[5]
'June'
>>> months.append("July")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    months.append("July")
AttributeError: 'tuple' object has no attribute 'append'
>>> 
```

STEP 3

You can create grouped tuples into lists that contain multiple sets of data. For instance, here is a tuple called NPC (Non-Playable Characters) containing the character name and their combat rating for an adventure game:

```
NPC=[("Conan", 100), ("Belit", 80), ("Valeria",
95)]
```



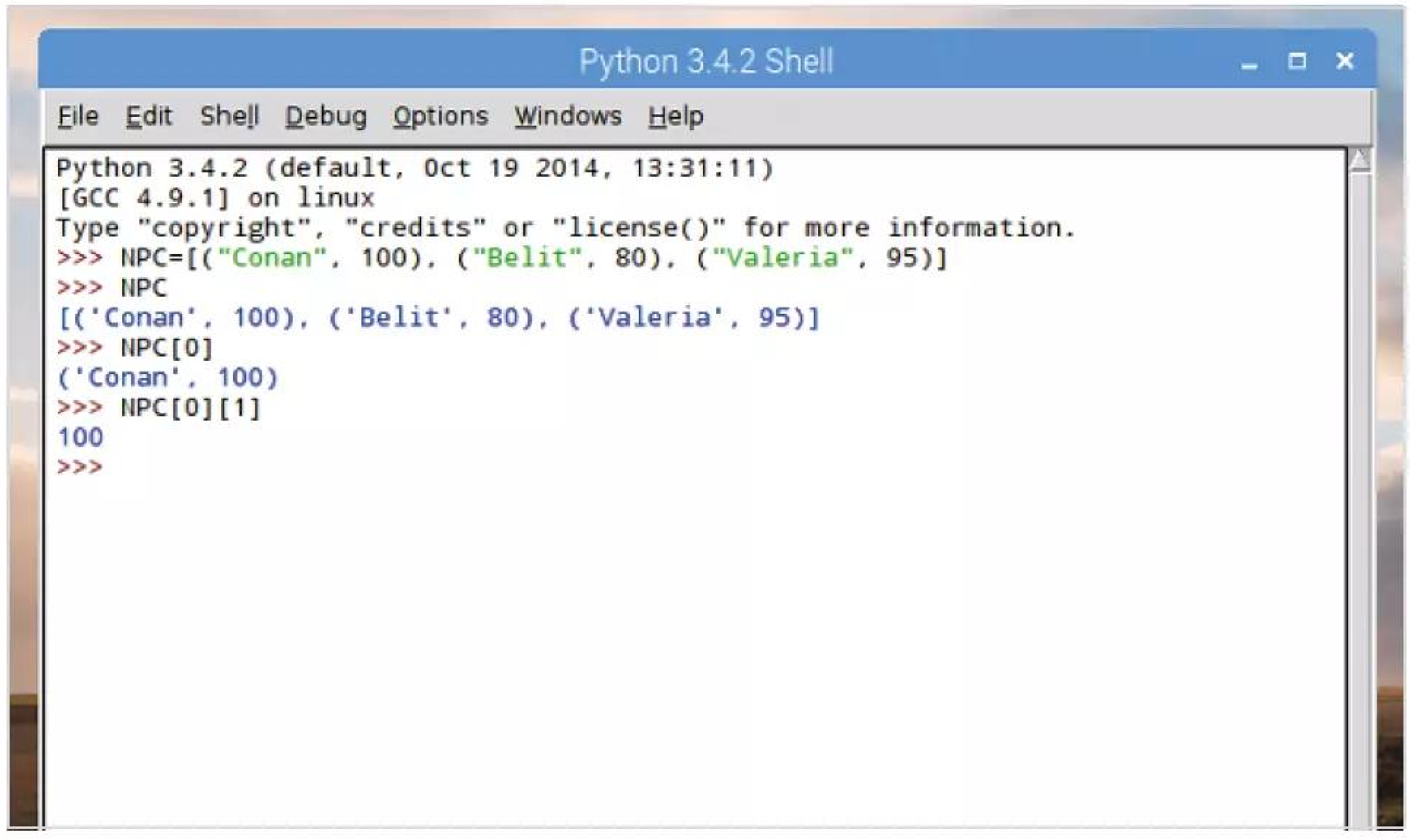
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan",100), ("Belit", 80), ("Valeria", 95)]
>>> 
```

STEP 4

Each of these data items can be accessed as a whole by entering NPC into the Shell; or they can be indexed according to their position NPC[0]. You can also index the individual tuples within the NPC list:

```
NPC[0][1]
```

Will display 100.

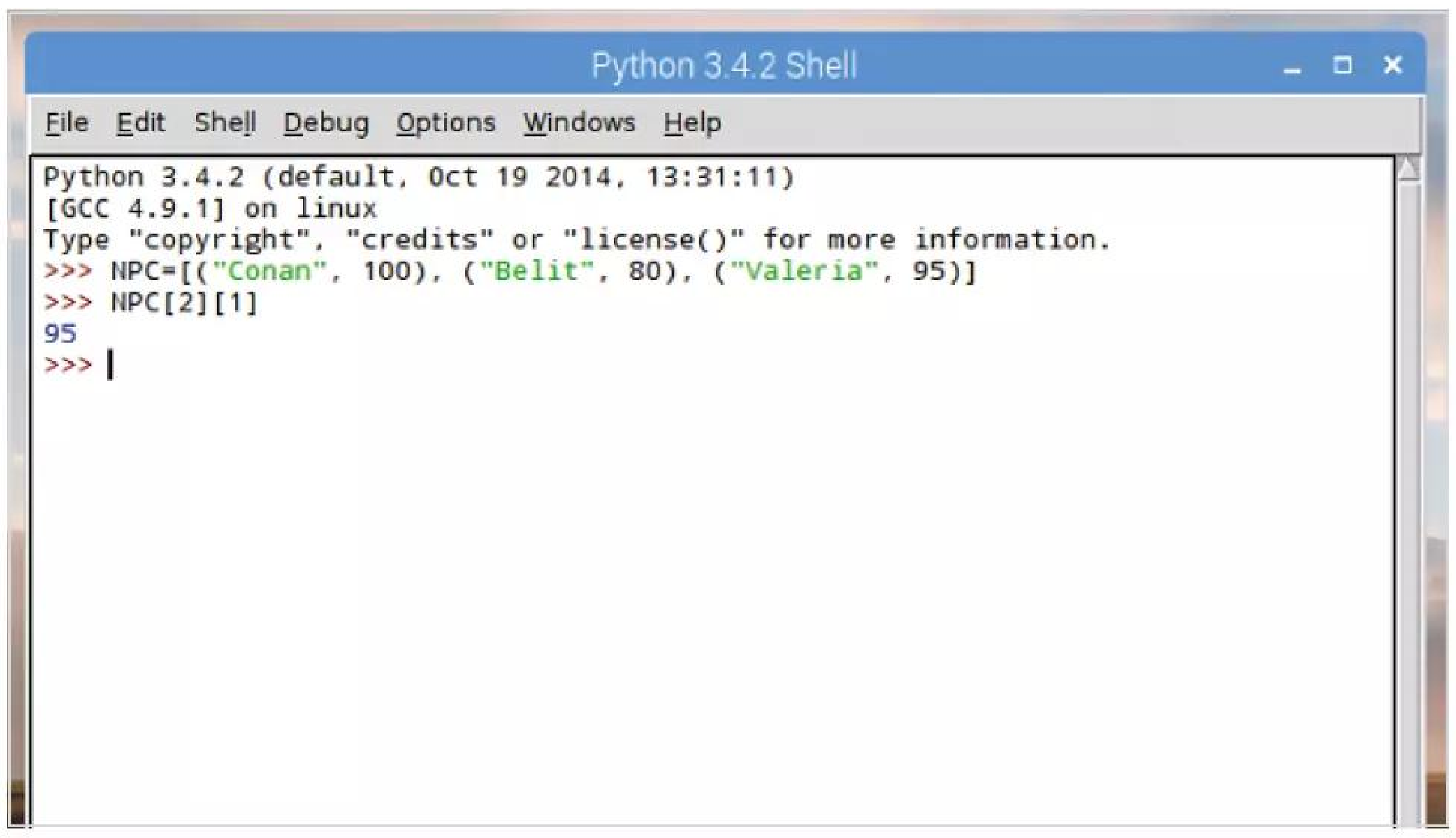


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC
[('Conan', 100), ('Belit', 80), ('Valeria', 95)]
>>> NPC[0]
('Conan', 100)
>>> NPC[0][1]
100
>>>
```

STEP 5

It's worth noting that when referencing multiple tuples within a list, the indexing is slightly different from the norm. You would expect the 95 combat rating of the character Valeria to be NPC[4][5], but it's not. It's actually:

`NPC[2][1]`


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

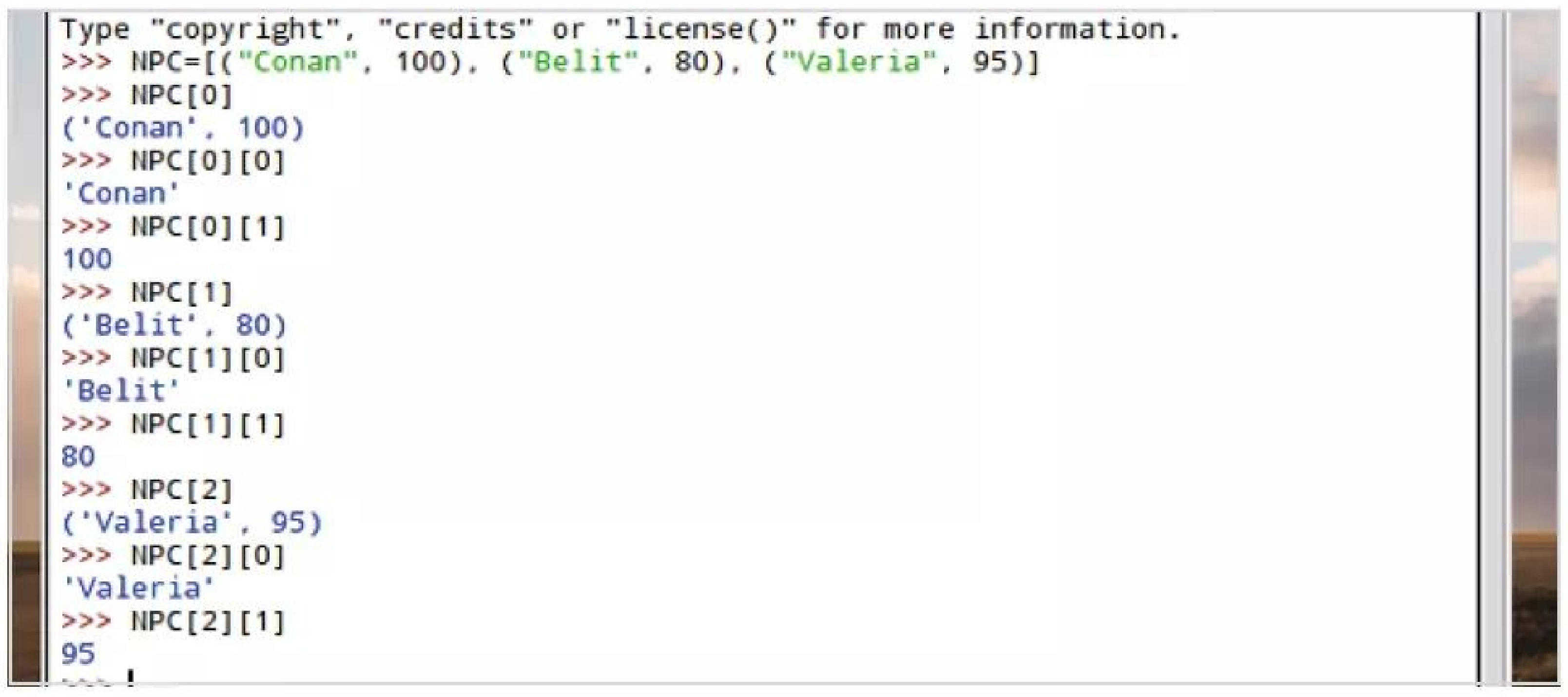
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC[2][1]
95
>>> |
```

STEP 6

This means of course that the indexing follows thus:

`0`
`0, 0`
`0, 1`
`1`
`1, 0`
`1, 1`
`2`
`2, 0`
`2,1`

Which as you can imagine, gets a little confusing when you've got a lot of tuple data to deal with.

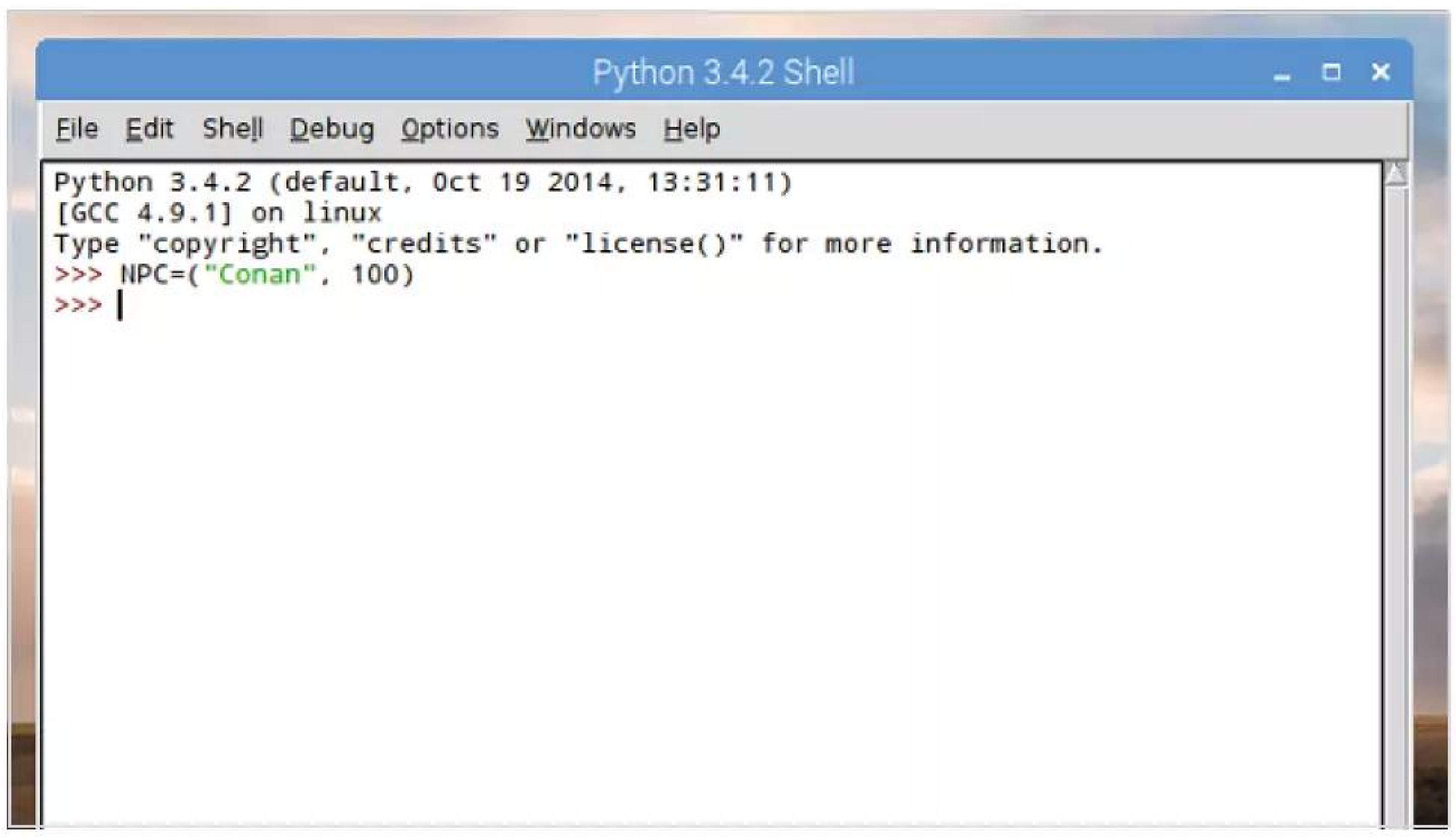


```
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC[0]
('Conan', 100)
>>> NPC[0][0]
'Conan'
>>> NPC[0][1]
100
>>> NPC[1]
('Belit', 80)
>>> NPC[1][0]
'Belit'
>>> NPC[1][1]
80
>>> NPC[2]
('Valeria', 95)
>>> NPC[2][0]
'Valeria'
>>> NPC[2][1]
95
```

STEP 7

Tuples though utilise a feature called unpacking, where the data items stored within a tuple are assigned variables. First create the tuple with two items (name and combat rating):

`NPC=("Conan", 100)`


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

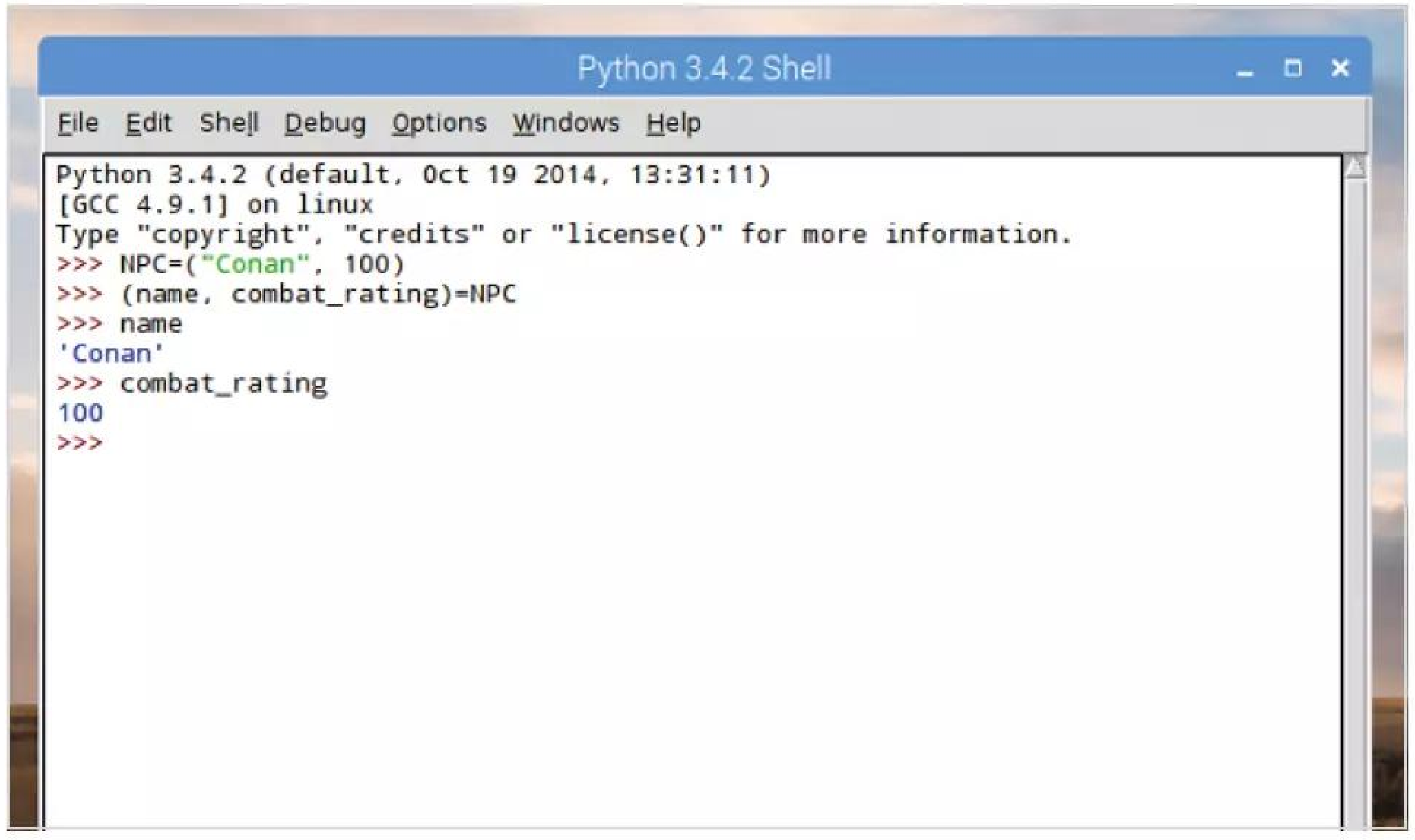
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=("Conan", 100)
>>> |
```

STEP 8

Now unpack the tuple into two corresponding variables:

`(name, combat_rating)=NPC`

You can now check the values by entering name and combat_rating.

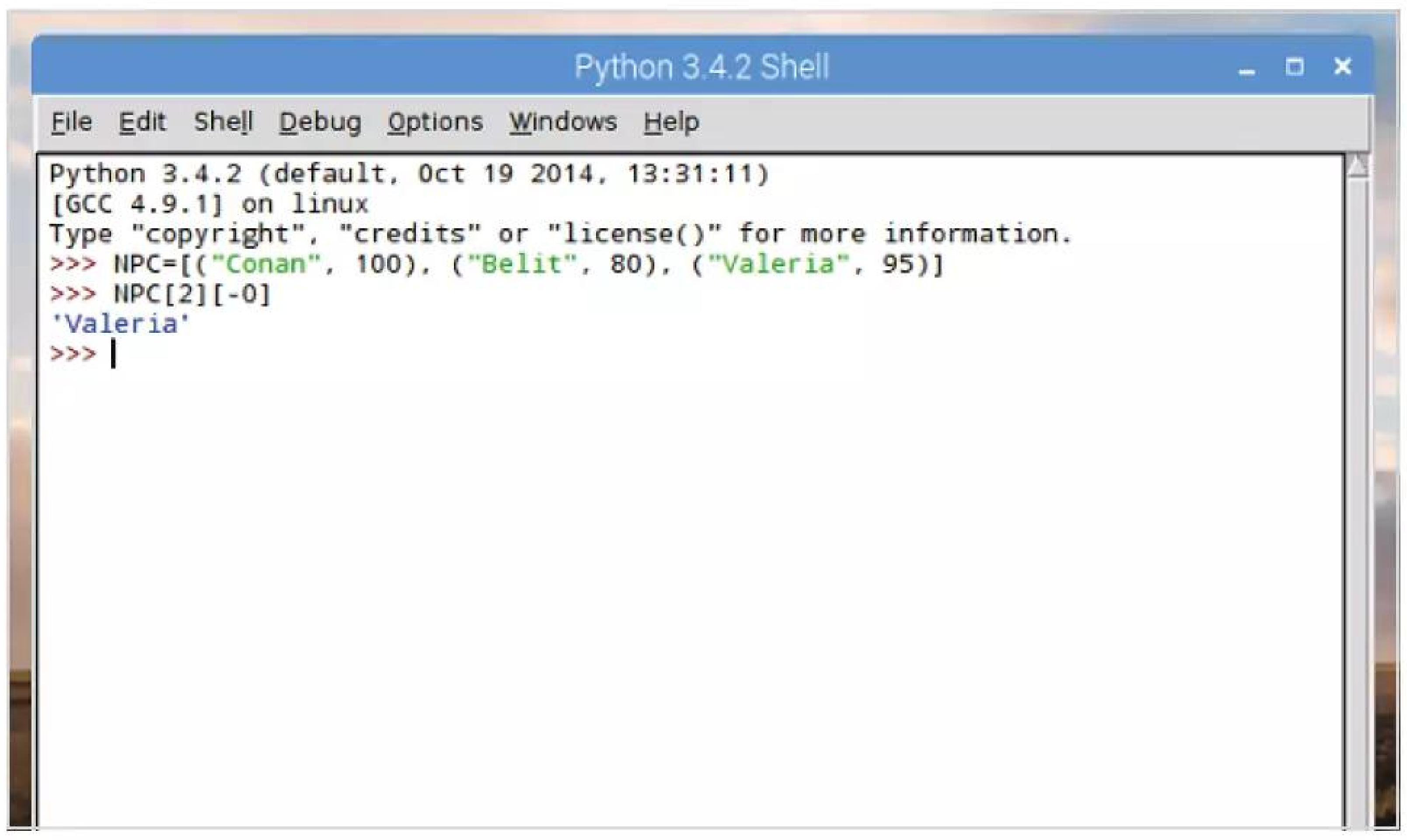

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=("Conan", 100)
>>> (name, combat_rating)=NPC
>>> name
'Conan'
>>> combat_rating
100
>>>
```

STEP 9

Remember, as with lists, you can also index tuples using negative numbers which count backwards from the end of the data list. For our example, using the tuple with multiple data items, you would reference the Valeria character with:

`NPC[2][-0]`


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> NPC=[("Conan", 100), ("Belit", 80), ("Valeria", 95)]
>>> NPC[2][-0]
'Valeria'
>>> |
```

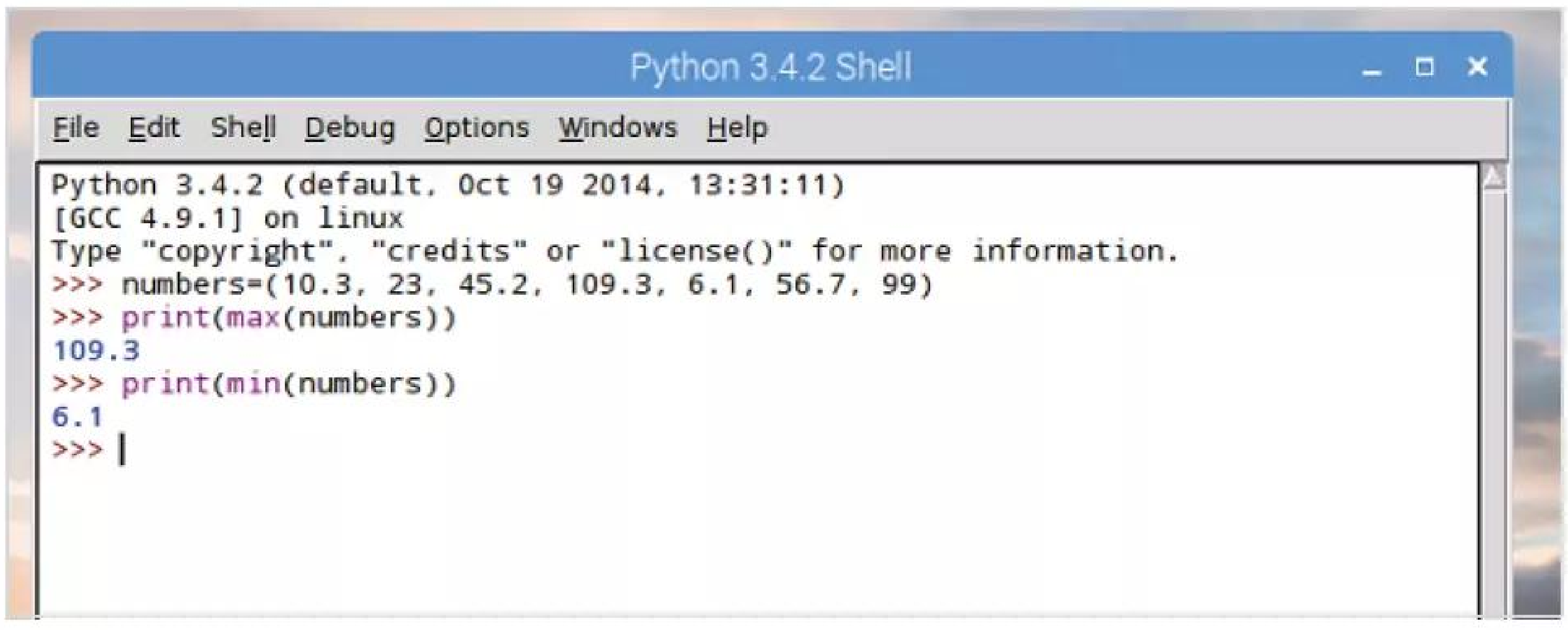
STEP 10

You can use the max and min functions to find the highest and lowest values of a tuple composed of numbers. For example:

`numbers=(10.3, 23, 45.2, 109.3, 6.1, 56.7, 99)`

The numbers can be integers and floats. To output the highest and lowest, use:

```
print(max(numbers))
print(min(numbers))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers=(10.3, 23, 45.2, 109.3, 6.1, 56.7, 99)
>>> print(max(numbers))
109.3
>>> print(min(numbers))
6.1
>>> |
```

Dictionaries

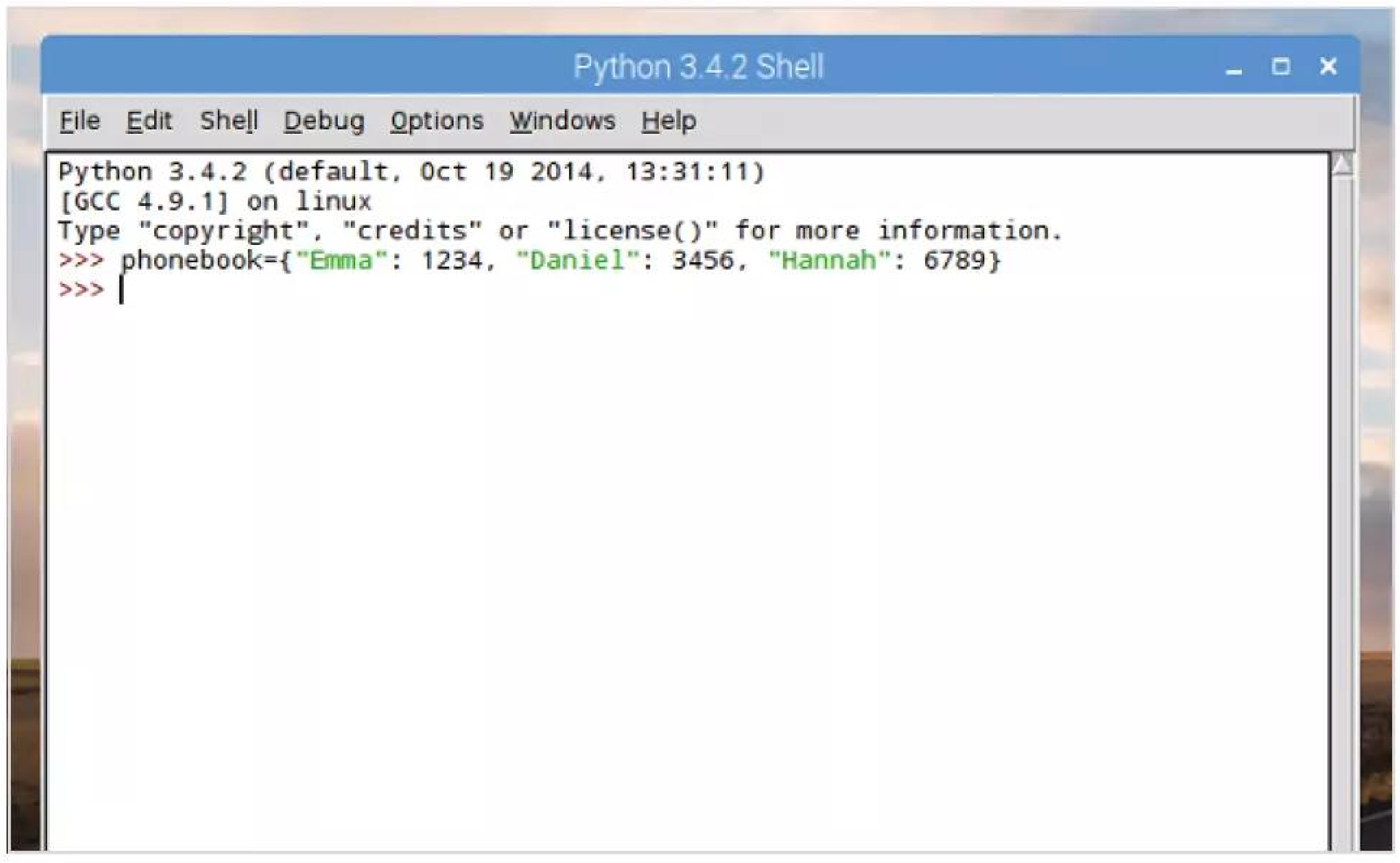
Lists are extremely useful but dictionaries in Python are by far the more technical way of dealing with data items. They can be tricky to get to grips with at first but you'll soon be able to apply them to your own code.

KEY PAIRS

A dictionary is like a list but instead each data item comes as a pair, these are known as Key and Value. The Key part must be unique and can either be a number or string whereas the Value can be any data item you like.

STEP 1 Let's say you want to create a phonebook in Python. You would create the dictionary name and enter the data in curly brackets, separating the key and value by a colon `Key:Value`. For example:

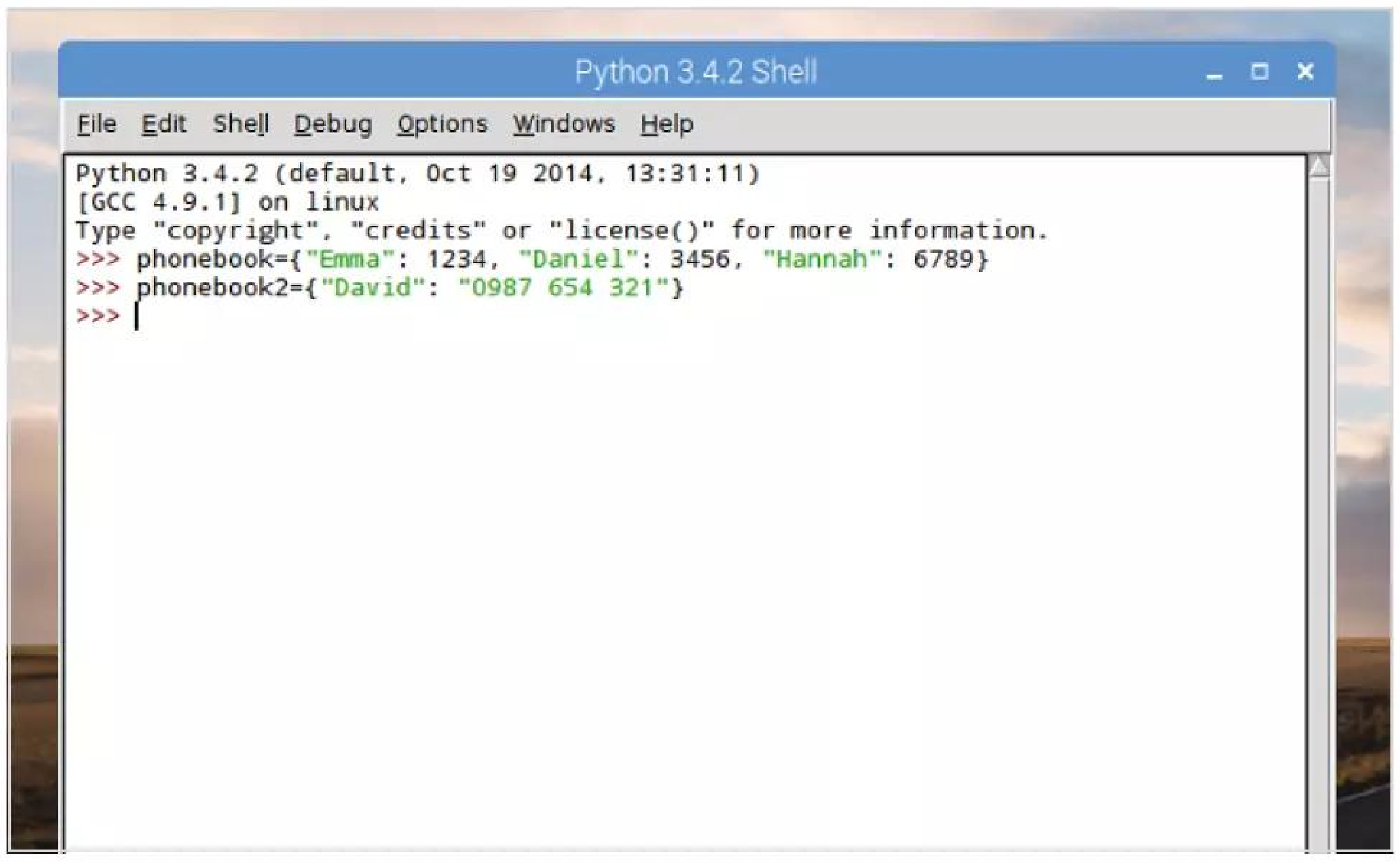
```
phonebook={“Emma”: 1234, “Daniel”: 3456, “Hannah”:
6789}
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> 
```

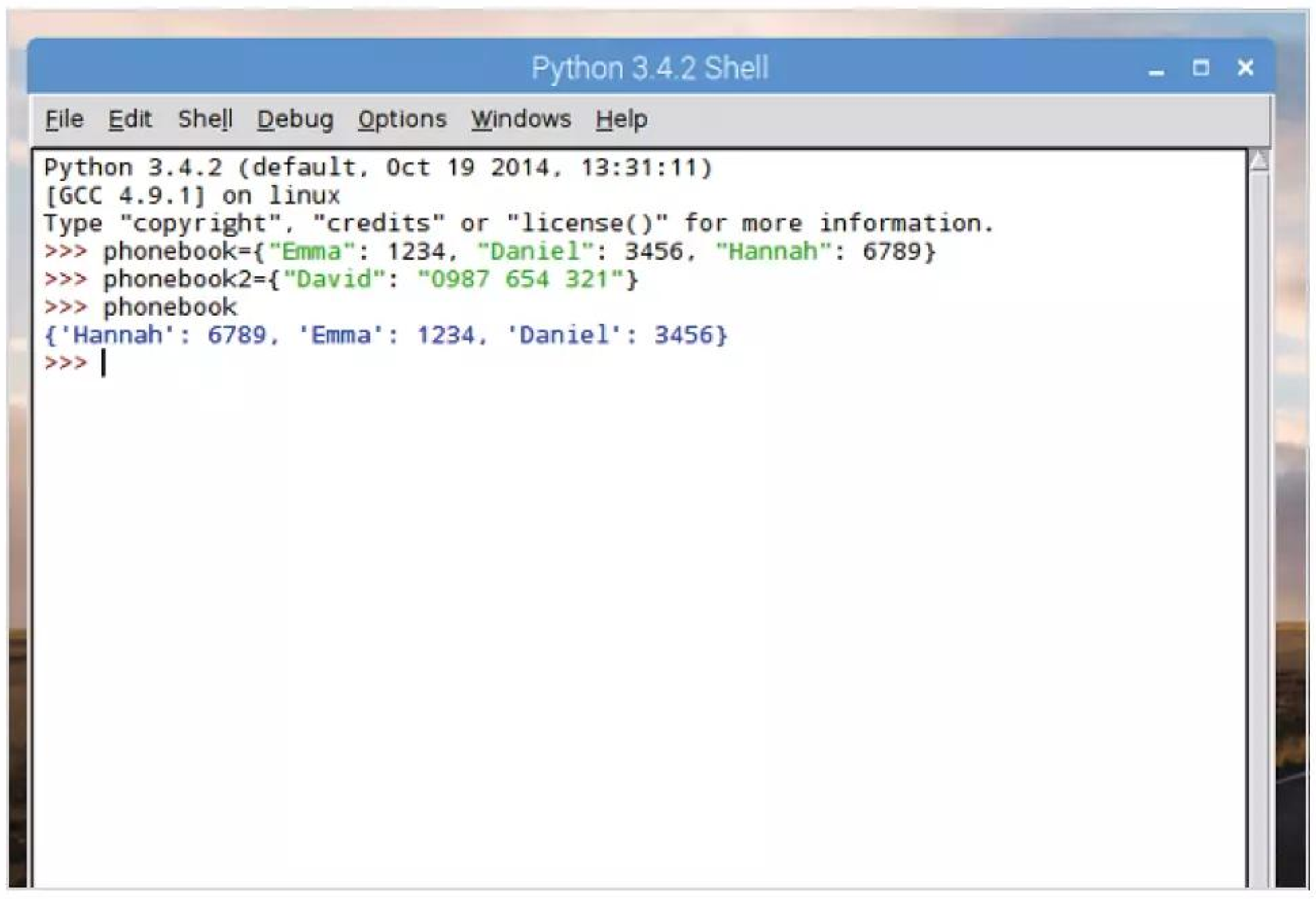
STEP 2 Just as with most lists, tuples and so on, strings need be enclosed in quotes (single or double), whilst integers can be left open. Remember that the value can be either a string or an integer, you just need to enclose the relevant one in quotes:

```
phonebook2={“David”: “0987 654 321”}
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> 
```

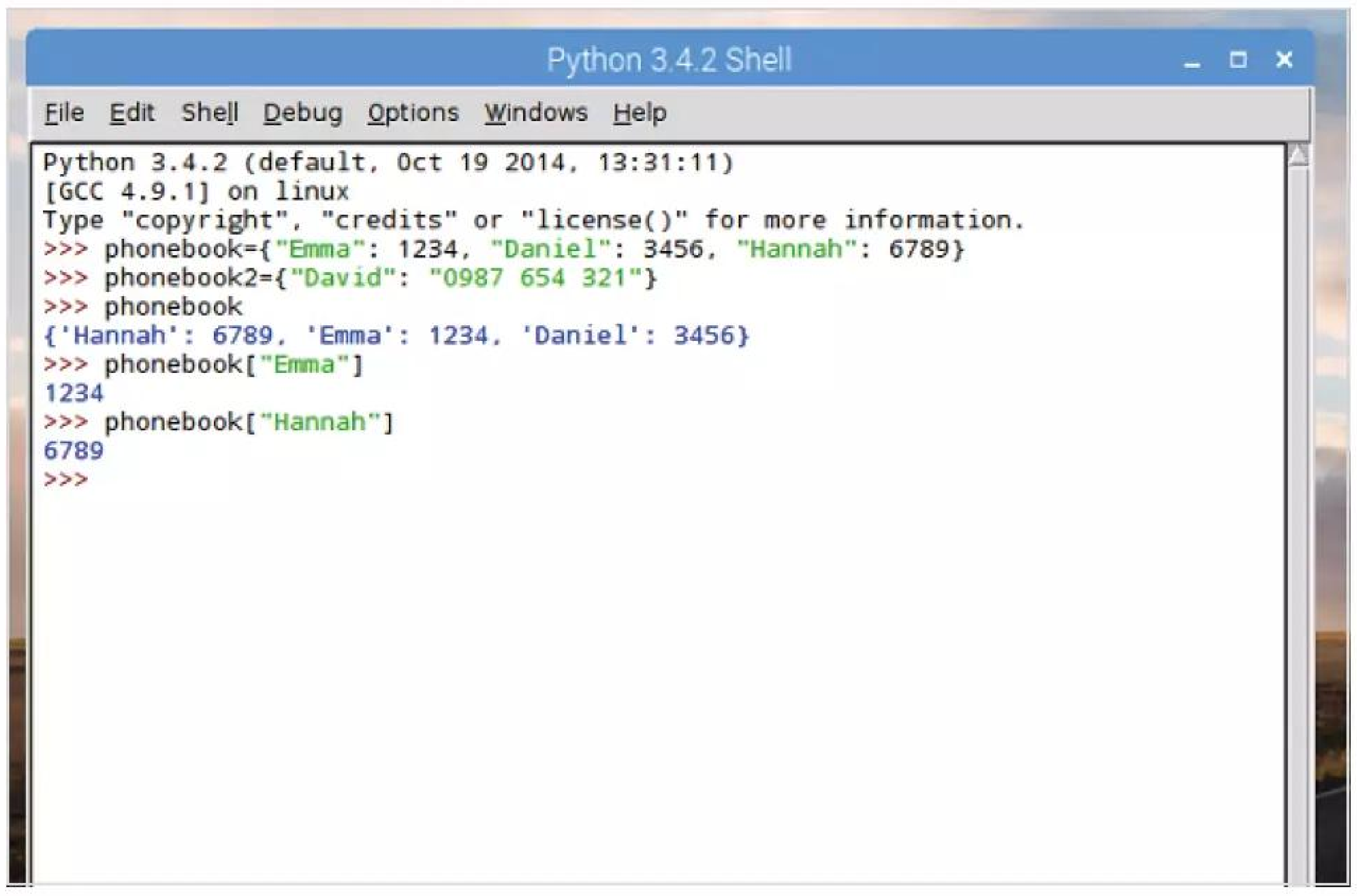
STEP 3 As with lists and tuples, you can check the contents of a dictionary by giving the dictionary a name: phonebook, in this example. This will display the data items you've entered in a similar fashion to a list, which you're no doubt familiar with by now.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> 
```

STEP 4 The benefit of using a dictionary is that you can enter the key to index the value. Using the phonebook example from the previous steps, you can enter:

```
phonebook[“Emma”]
phonebook[“Hannah”]
```

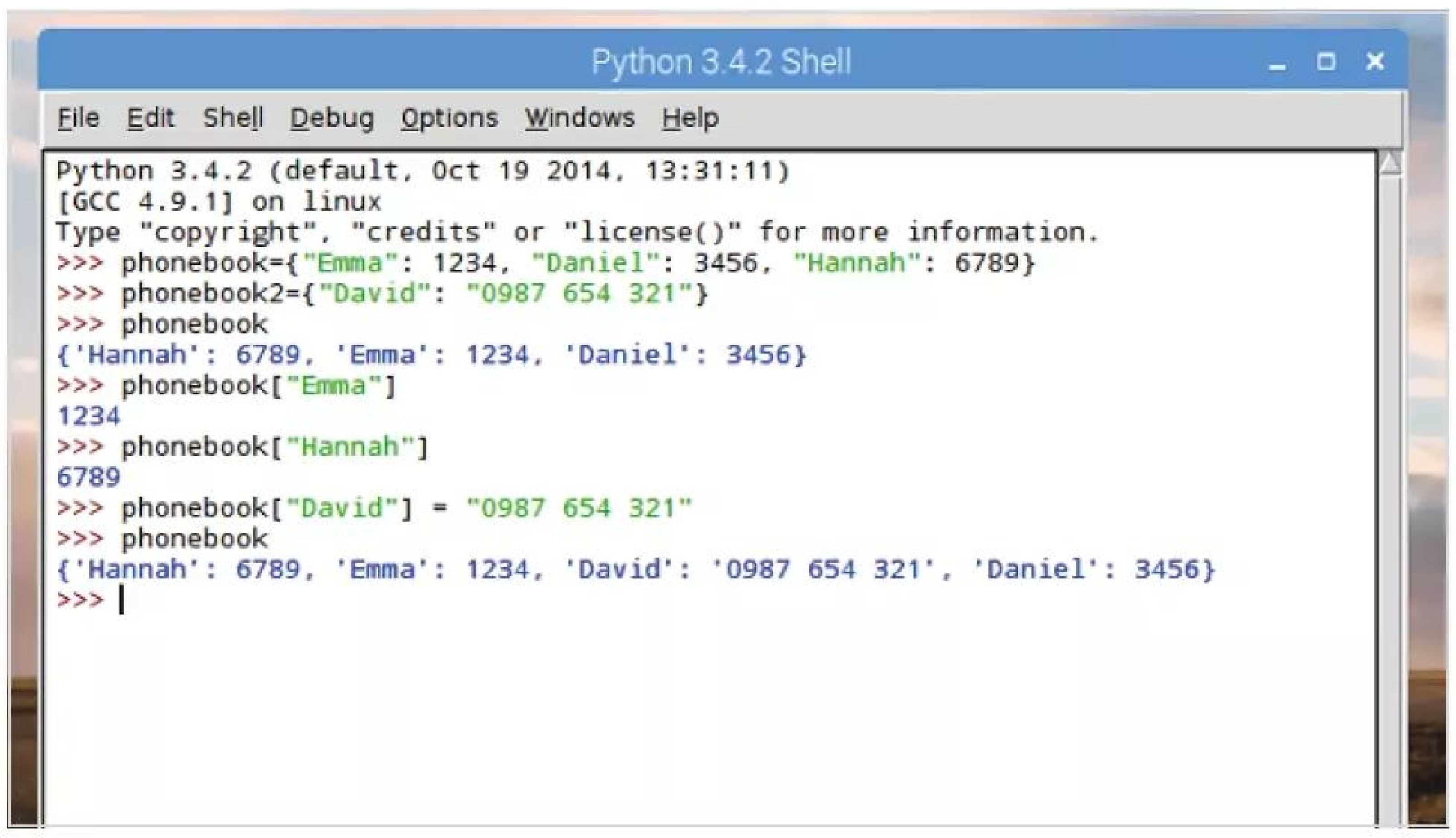


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> 
```

STEP 5

Adding to a dictionary is easy too. You can include a new data item entry by adding the new key and value items like:

```
phonebook["David"] = "0987 654 321"
phonebook
```

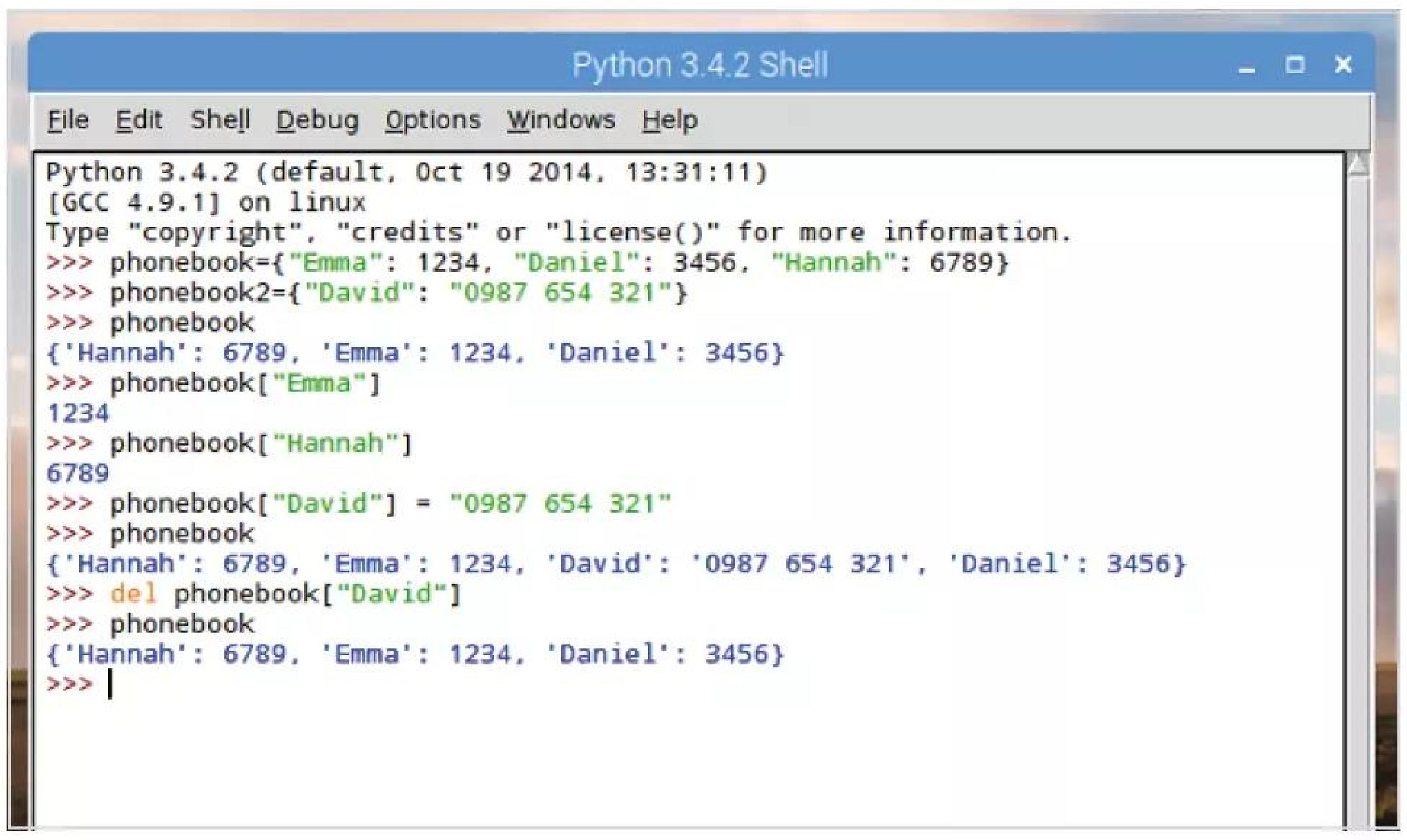

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> phonebook["David"] = "0987 654 321"
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'David': '0987 654 321', 'Daniel': 3456}
>>> |
```

STEP 6

You can also remove items from a dictionary by issuing the del command followed by the item's key; the value will be removed as well, since both work as a pair of data items:

```
del phonebook["David"]
```

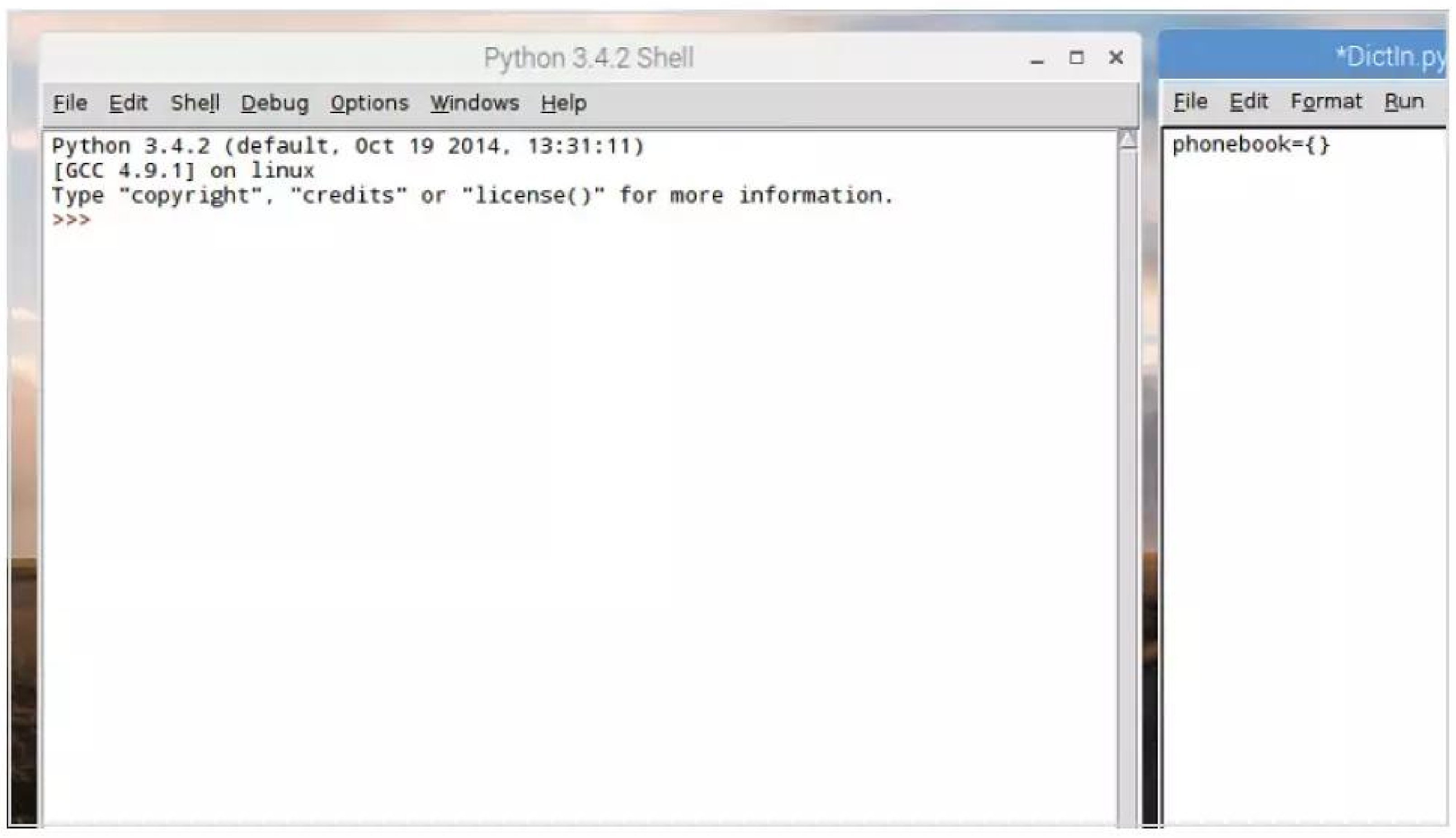

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> phonebook={"Emma": 1234, "Daniel": 3456, "Hannah": 6789}
>>> phonebook2={"David": "0987 654 321"}
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> phonebook["Emma"]
1234
>>> phonebook["Hannah"]
6789
>>> phonebook["David"] = "0987 654 321"
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'David': '0987 654 321', 'Daniel': 3456}
>>> del phonebook["David"]
>>> phonebook
{'Hannah': 6789, 'Emma': 1234, 'Daniel': 3456}
>>> |
```

STEP 7

Taking this a step further, how about creating a piece of code that will ask the user for the dictionary key and value items? Create a new Editor instance and start by coding in a new, blank dictionary:

```
phonebook={}
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
*DictIn.py
File Edit Format Run

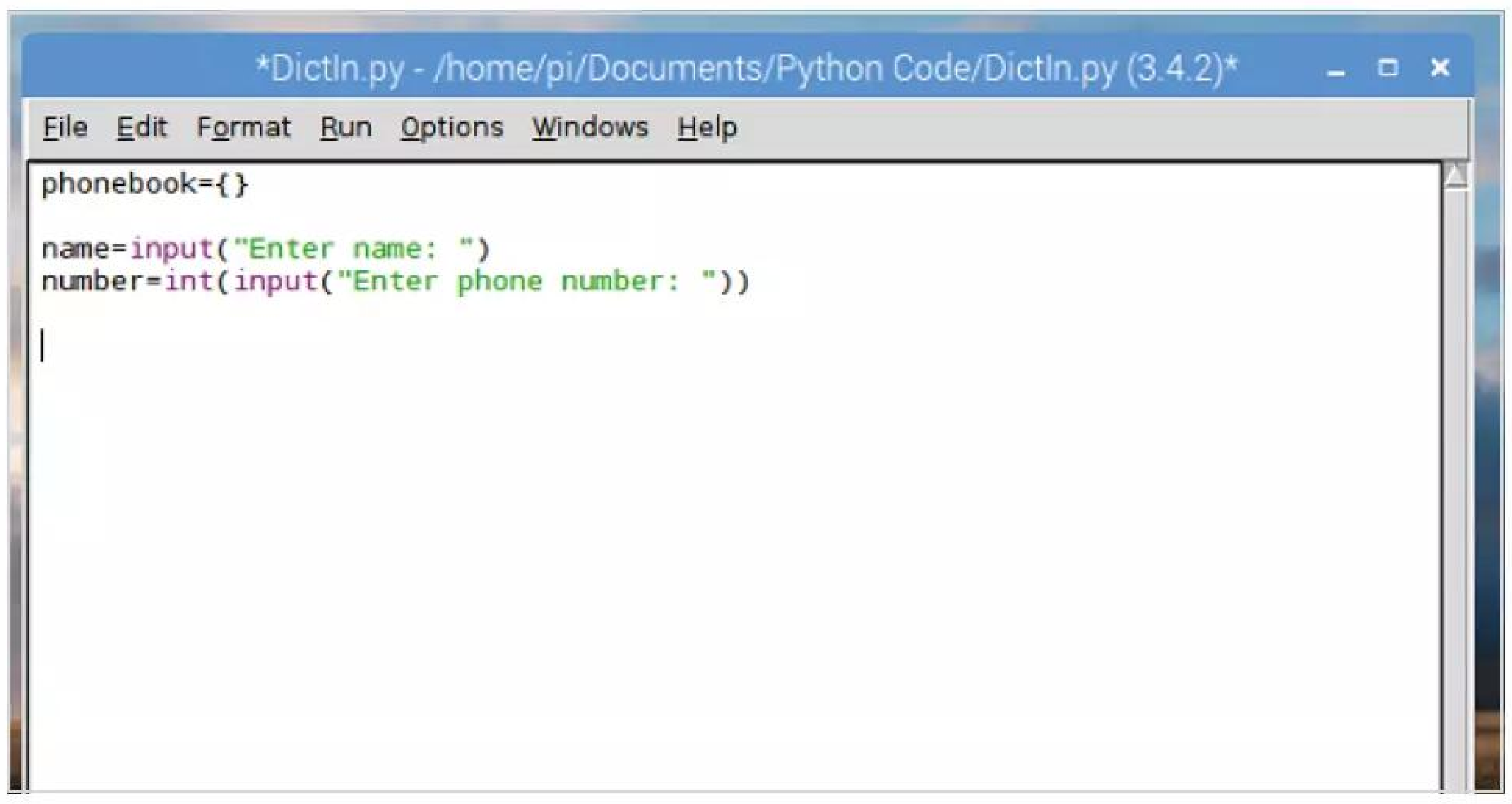
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
phonebook={}
```

STEP 8

Next, you need to define the user inputs and variables: one for the person's name, the other for their phone number (let's keep it simple to avoid lengthy Python code):

```
name=input("Enter name: ")
number=int(input("Enter phone number: "))
```

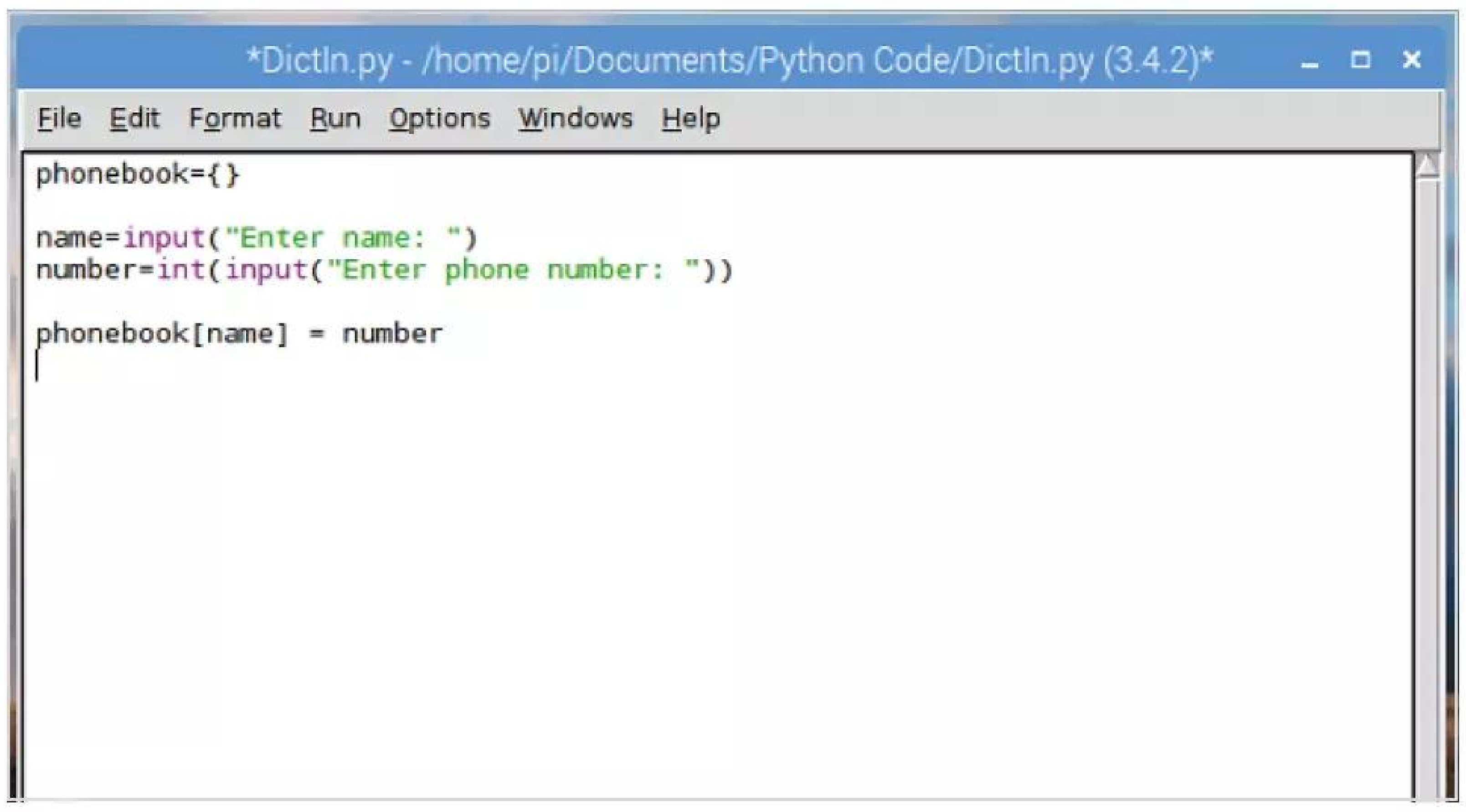

DictIn.py - /home/pi/Documents/Python Code/DictIn.py (3.4.2)
File Edit Format Run Options Windows Help

```
phonebook={}

name=input("Enter name: ")
number=int(input("Enter phone number: "))

|
```

STEP 9

Note we've kept the number as an integer instead of a string, even though the value can be both an integer or a string. Now you need to add the user's inputted variables to the newly created blank dictionary. Using the same process as in Step 5, you can enter:

```
phonebook[name] = number
```


DictIn.py - /home/pi/Documents/Python Code/DictIn.py (3.4.2)
File Edit Format Run Options Windows Help

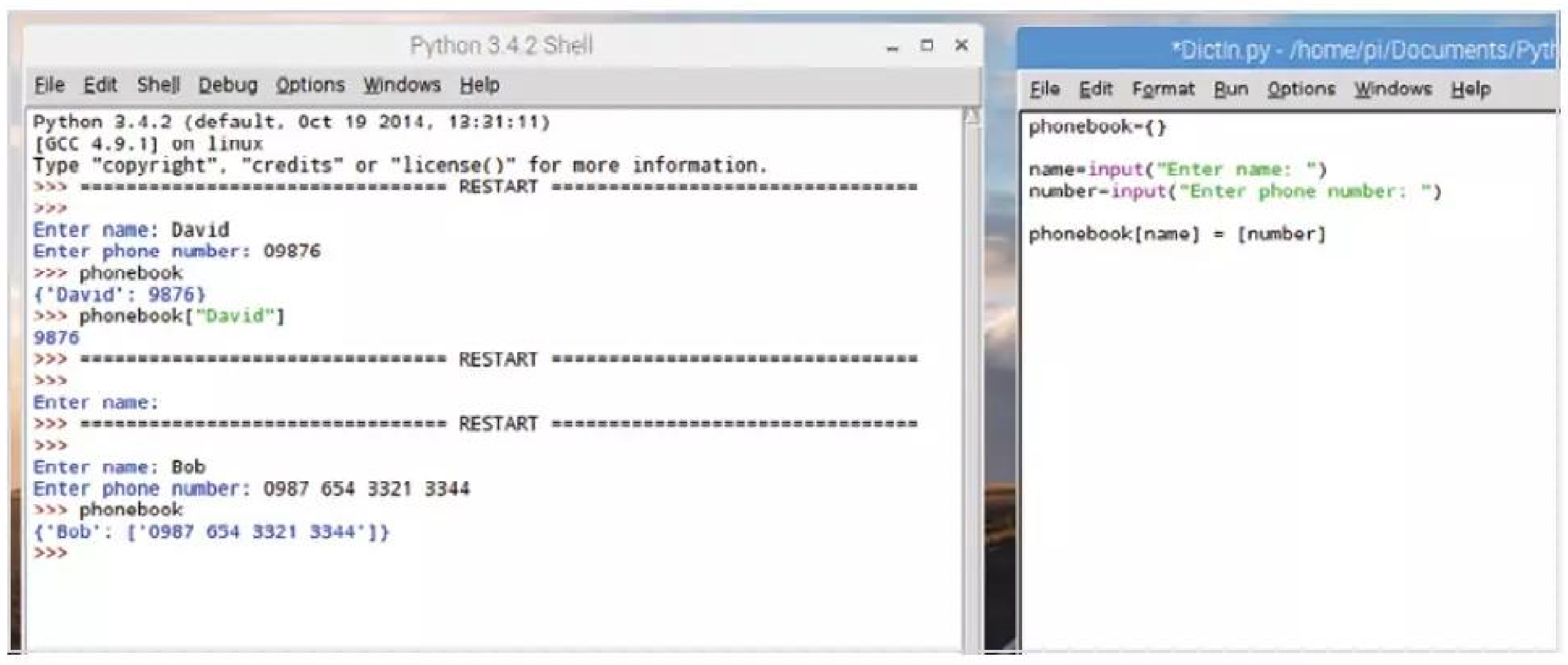
```
phonebook={}

name=input("Enter name: ")
number=int(input("Enter phone number: "))

phonebook[name] = number
|
```

STEP 10

Now when you save and execute the code, Python will ask for a name and a number. It will then insert those entries into the phonebook dictionary, which you can test by entering into the Shell:

```
phonebook
phonebook["David"]
```

If the number needs to contain spaces you need to make it a string, so remove the int part of the input.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
*DictIn.py - /home/pi/Documents/Pyt
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter name: David
Enter phone number: 09876
>>> phonebook
{'David': 9876}
>>> phonebook["David"]
9876
>>> ================================ RESTART ================================
>>>
Enter name:
>>> ================================ RESTART ================================
>>>
Enter name: Bob
Enter phone number: 0987 654 3321 3344
>>> phonebook
{'Bob': ['0987 654 3321 3344']}
>>>
```

```
phonebook={}

name=input("Enter name: ")
number=input("Enter phone number: ")

phonebook[name] = [number]
```

Splitting and Joining Strings

When dealing with data in Python, especially from a user's input, you will undoubtedly come across long sets of strings. A useful skill to learn in Python programming is being able to split those long strings for better readability.

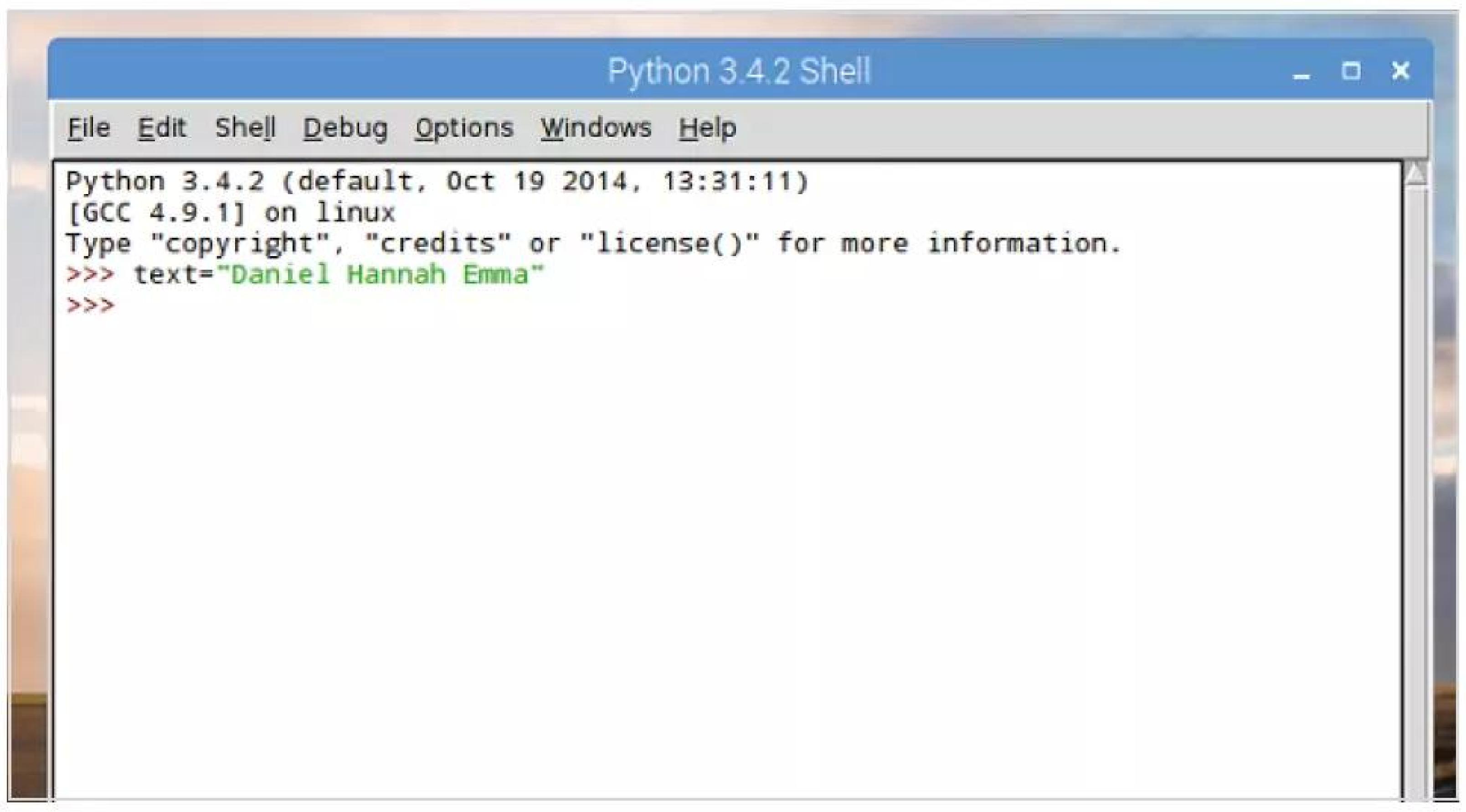
STRING THEORIES

You've already looked at some list functions, using .insert, .remove, and .pop but there are also functions that can be applied to strings.

STEP 1

The main tool in the string function arsenal is .split(). With it you're able to split apart a string of data, based on the argument within the brackets. For example, here's a string with three items, each separated by a space:

```
text="Daniel Hannah Emma"
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

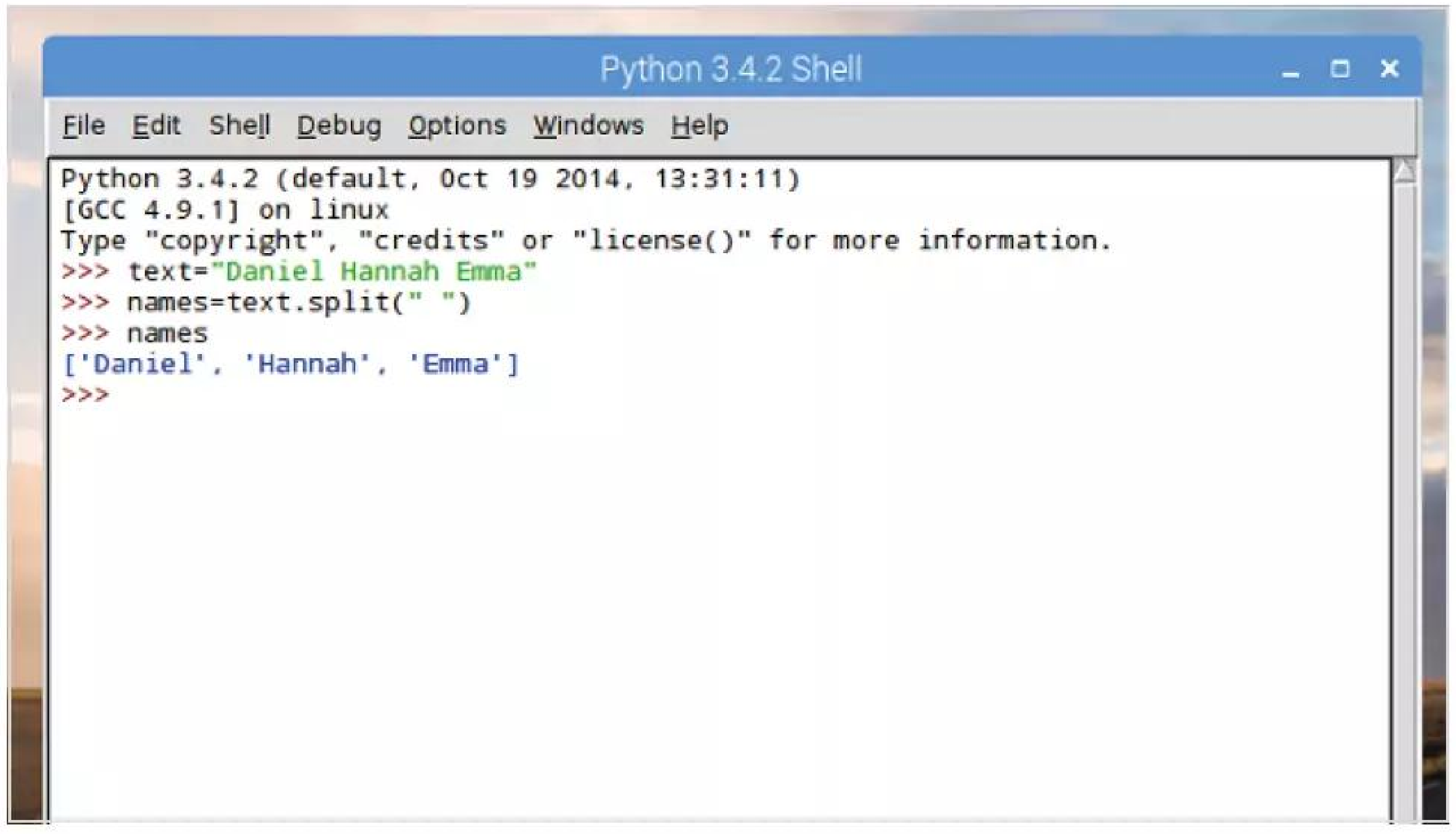
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> text="Daniel Hannah Emma"
>>>
```

STEP 2

Now let's turn the string into a list and split the content accordingly:

```
names=text.split(" ")
```

Then enter the name of the new list, names, to see the three items.

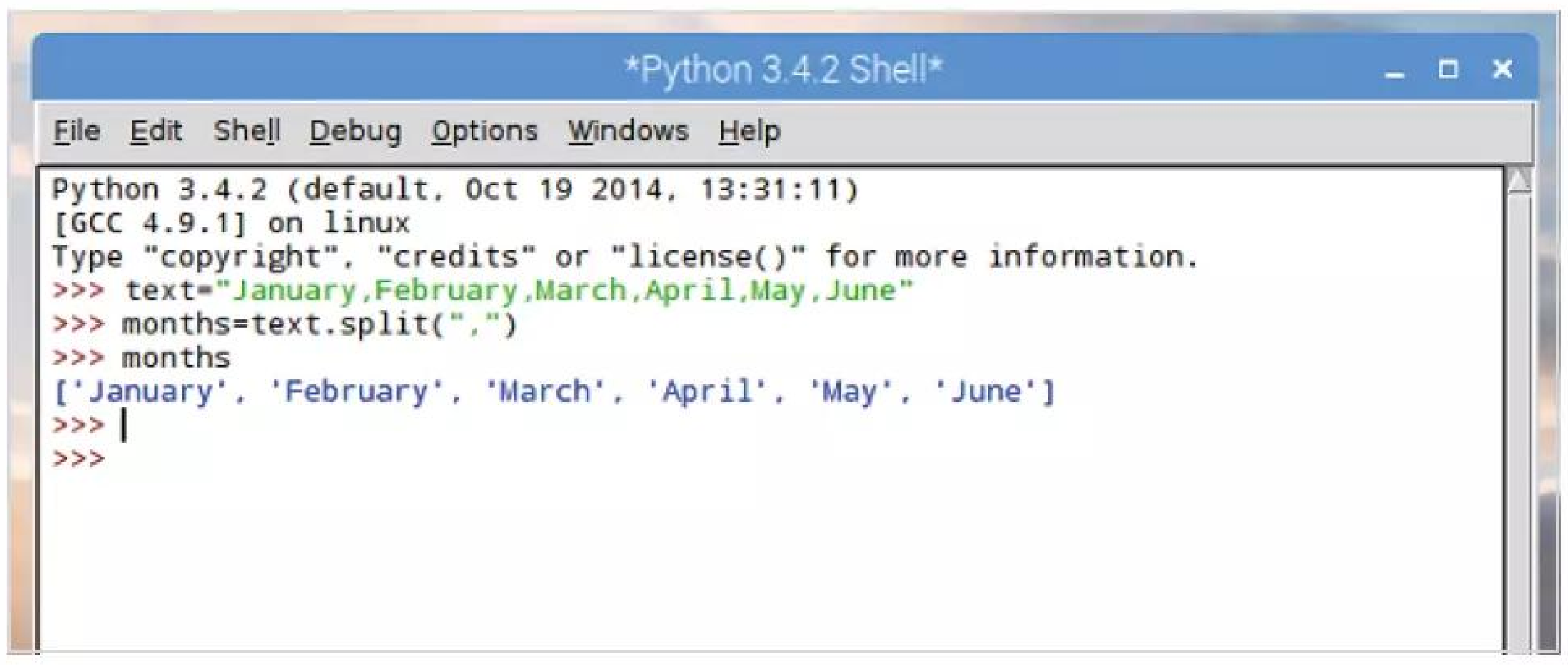

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> text="Daniel Hannah Emma"
>>> names=text.split(" ")
>>> names
['Daniel', 'Hannah', 'Emma']
>>>
```

STEP 3

Note that the text.split part has the brackets, quotes, then a space followed by closing quotes and brackets. The space is the separator, indicating that each list item entry is separated by a space. Likewise, CSV (Comma Separated Value) content has a comma, so you'd use:

```
text="January,February,March,April,May,June"
months=text.split(",")
months
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

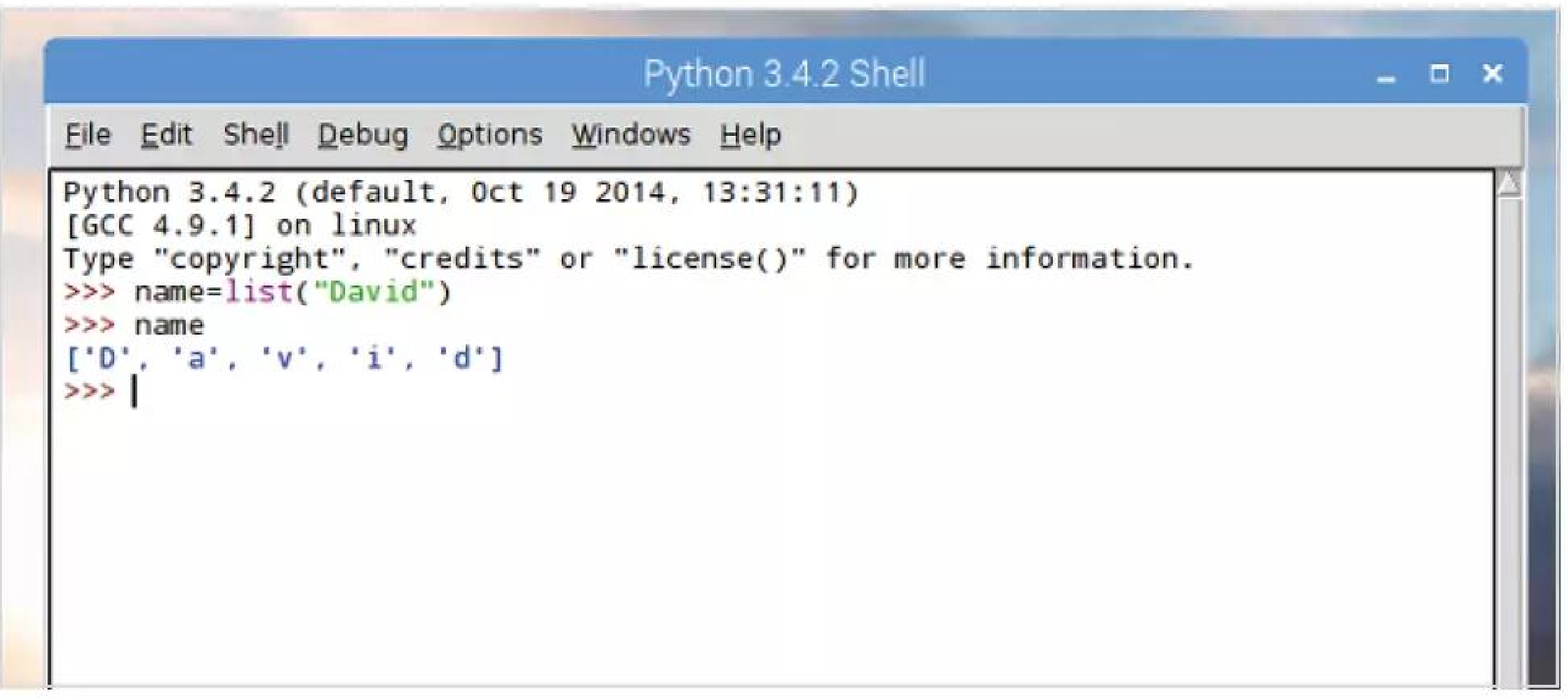
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> text="January,February,March,April,May,June"
>>> months=text.split(",")
>>> months
['January', 'February', 'March', 'April', 'May', 'June']
>>> 
>>>
```

STEP 4

You've previously seen how you can split a string into individual letters as a list, using a name:

```
name=list("David")
name
```

The returned value is 'D', 'a', 'v', 'i', 'd'. Whilst it may seem a little useless under ordinary circumstances, it could be handy for creating a spelling game for example.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

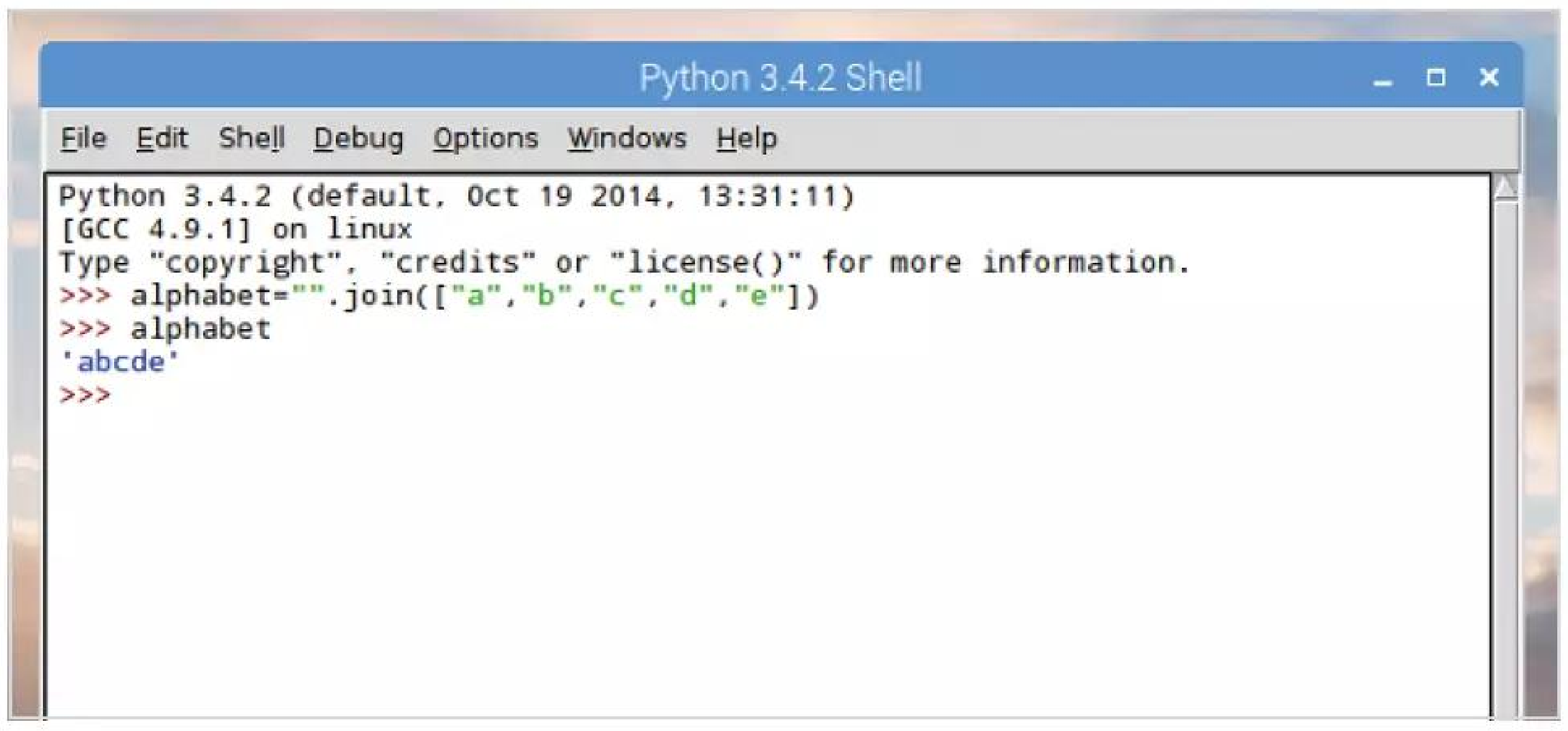
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name=list("David")
>>> name
['D', 'a', 'v', 'i', 'd']
>>> 
```

STEP 5

The opposite of the .split function is .join, where you will have separate items in a string and can join them all together to form a word or just a combination of items, depending on the program you're writing. For instance:

```
alphabet="".join([“a”,”b”,”c”,”d”,”e”])
alphabet
```

This will display 'abcde' in the Shell.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

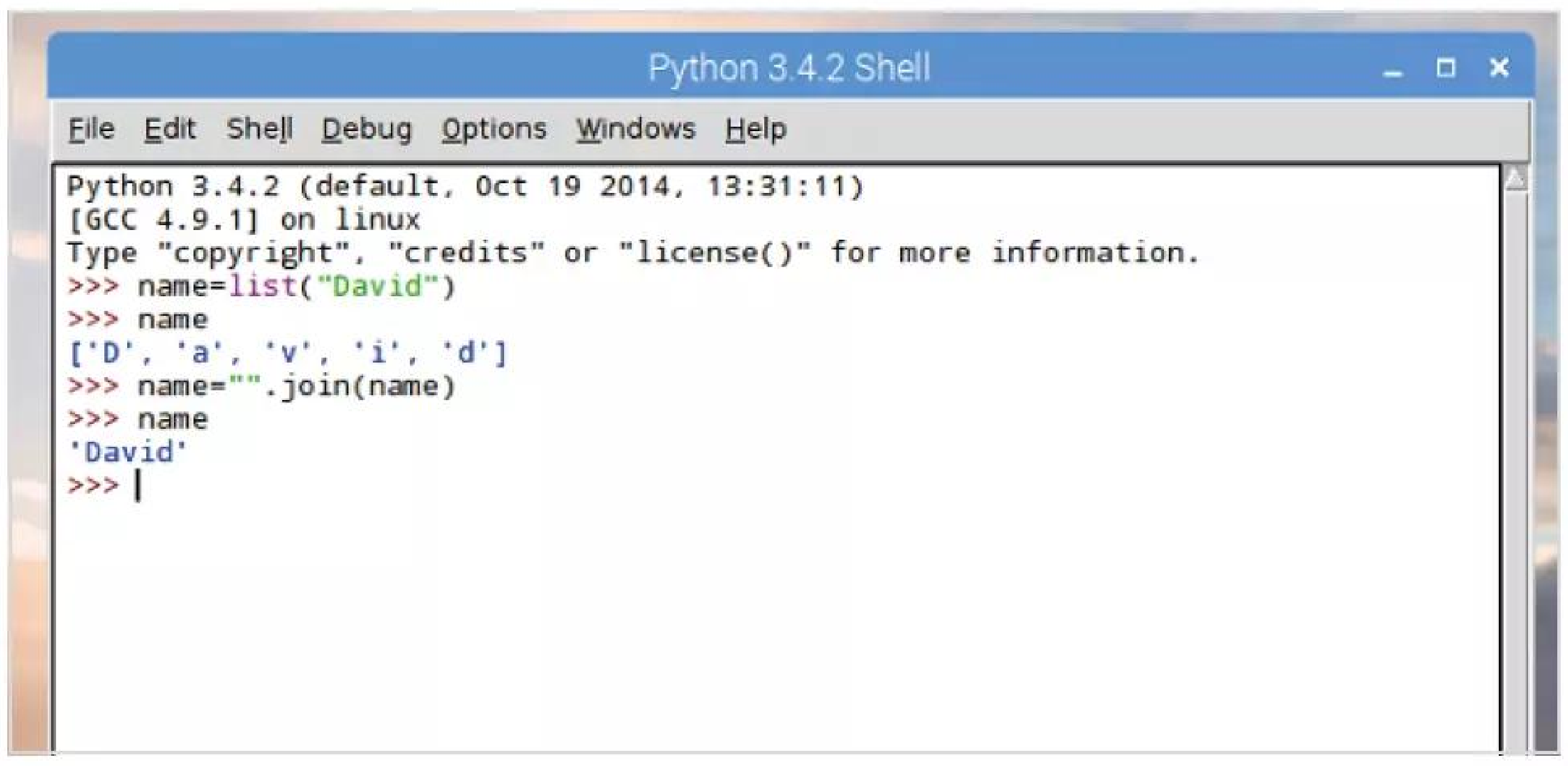
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> alphabet="".join(["a","b","c","d","e"])
>>> alphabet
'abcde'
>>>
```

STEP 6

You can therefore apply .join to the separated name you made in Step 4, combining the letters again to form the name:

```
name="".join(name)
name
```

We've joined the string back together, and retained the list called name, passing it through the .join function.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

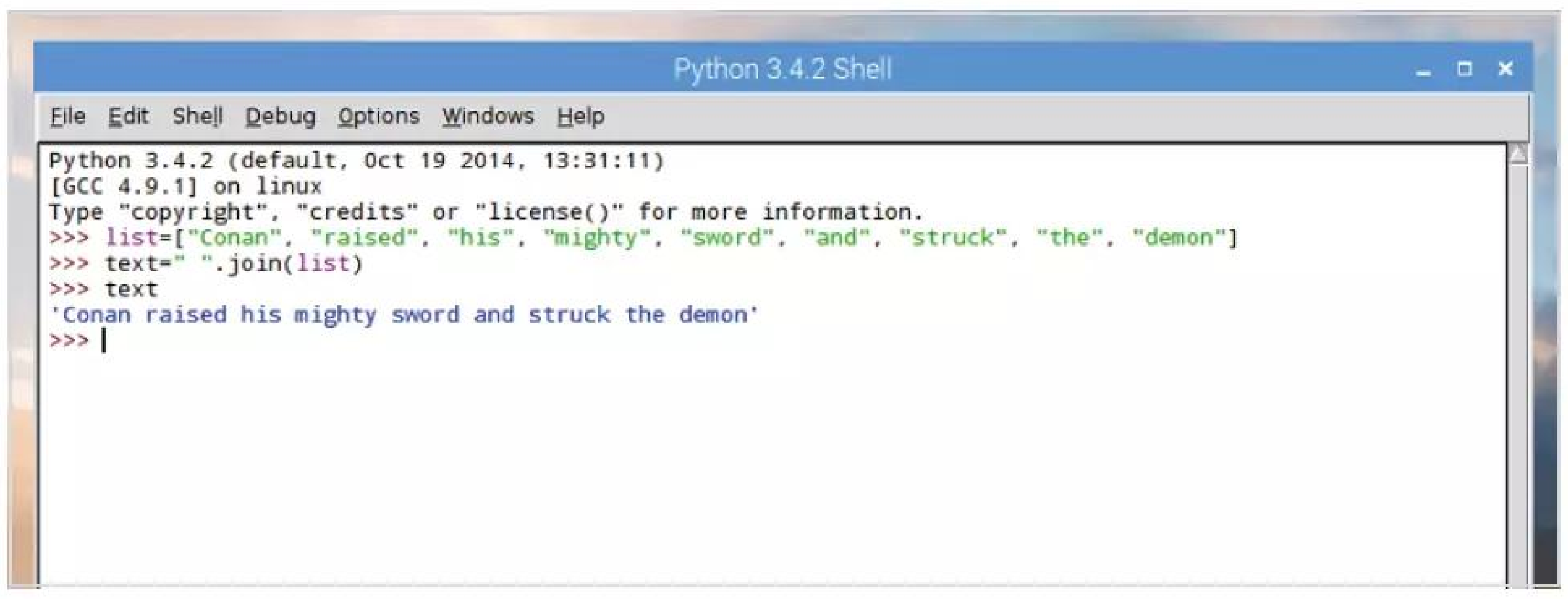
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name=list("David")
>>> name
['D', 'a', 'v', 'i', 'd']
>>> name="".join(name)
>>> name
'David'
>>> |
```

STEP 7

A good example of using the .join function is when you have a list of words you want to combine into a sentence:

```
list=[“Conan”, “raised”, “his”, “mighty”, “sword”,
“and”, “struck”, “the”, “demon”]
text=” “.join(list)
text
```

Note the space between the quotes before the .join function (where there were no spaces in the Step 6's join)

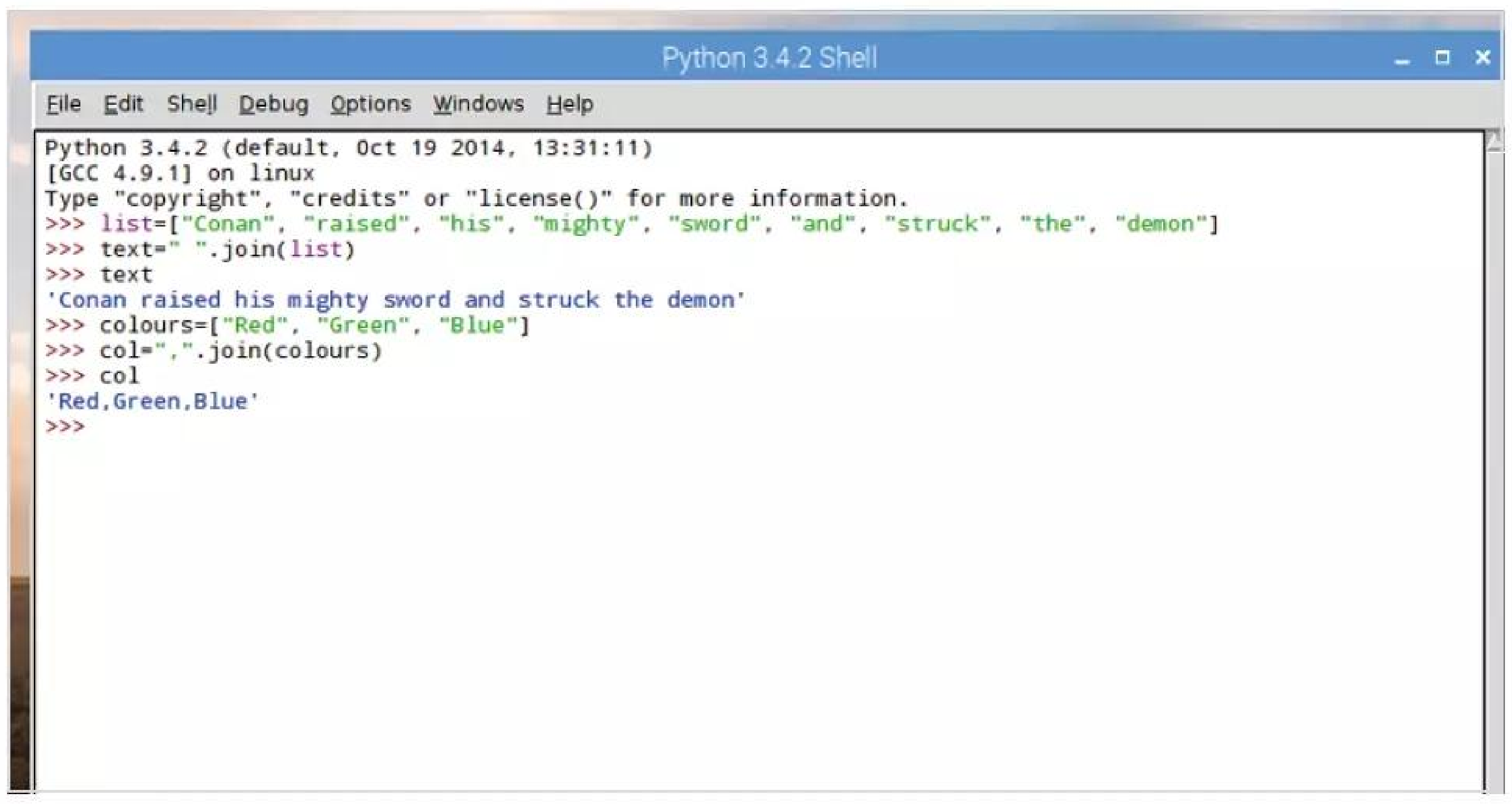

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list=["Conan", "raised", "his", "mighty", "sword", "and", "struck", "the", "demon"]
>>> text=" ".join(list)
>>> text
'Conan raised his mighty sword and struck the demon'
>>> |
```

STEP 8

As with the .split function, the separator doesn't have to be a space, it can also be a comma, a full stop, a hyphen or whatever you like:

```
colours=[“Red”, “Green”, “Blue”]
col=”,”.join(colours)
col
```

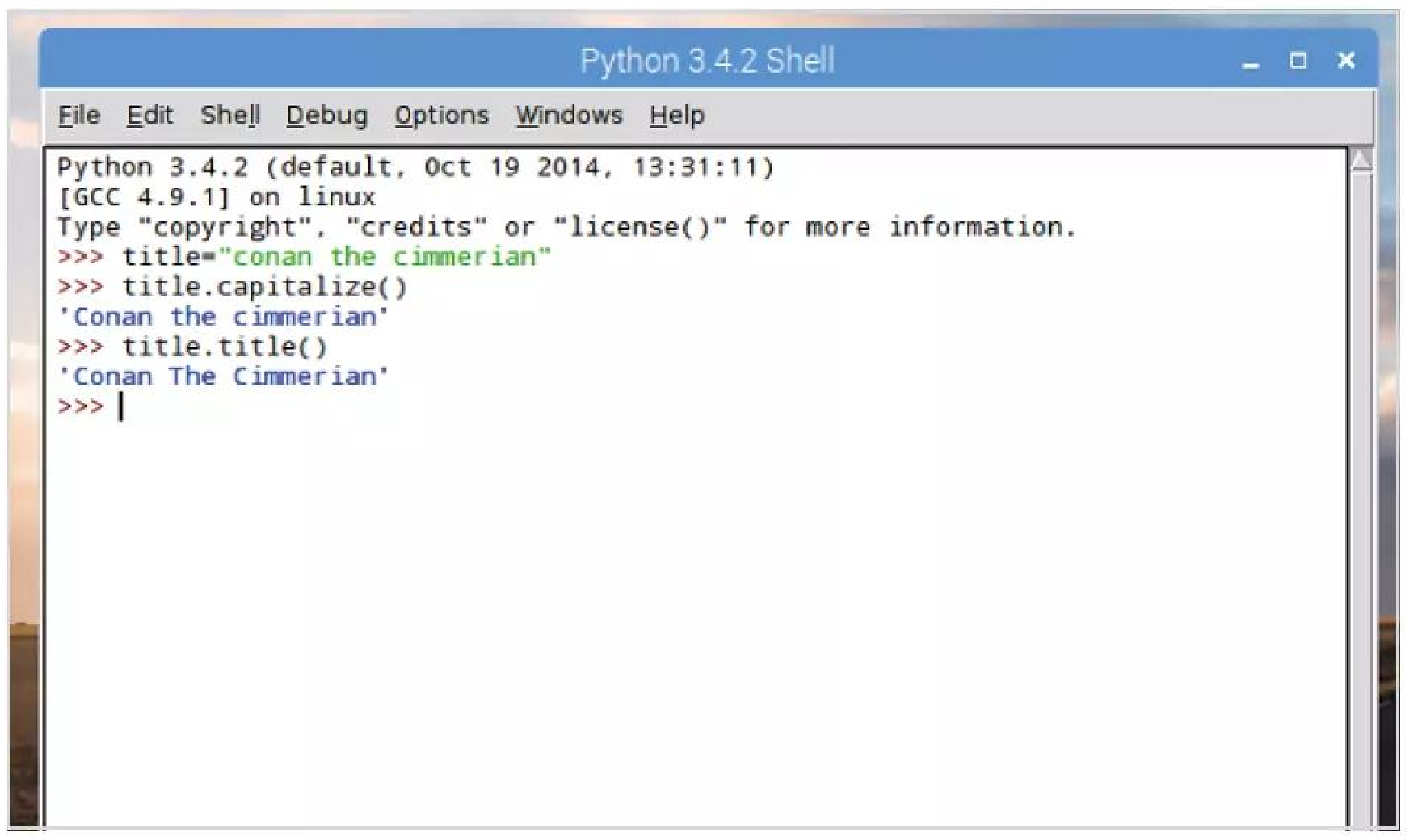

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> list=["Conan", "raised", "his", "mighty", "sword", "and", "struck", "the", "demon"]
>>> text=" ".join(list)
>>> text
'Conan raised his mighty sword and struck the demon'
>>> colours=["Red", "Green", "Blue"]
>>> col=",".join(colours)
>>> col
'Red,Green,Blue'
>>>
```

STEP 9

There's some interesting functions you apply to a string, such as .capitalize and .title. For example:

```
title=”conan the cimmerian”
title.capitalize()
title.title()
```

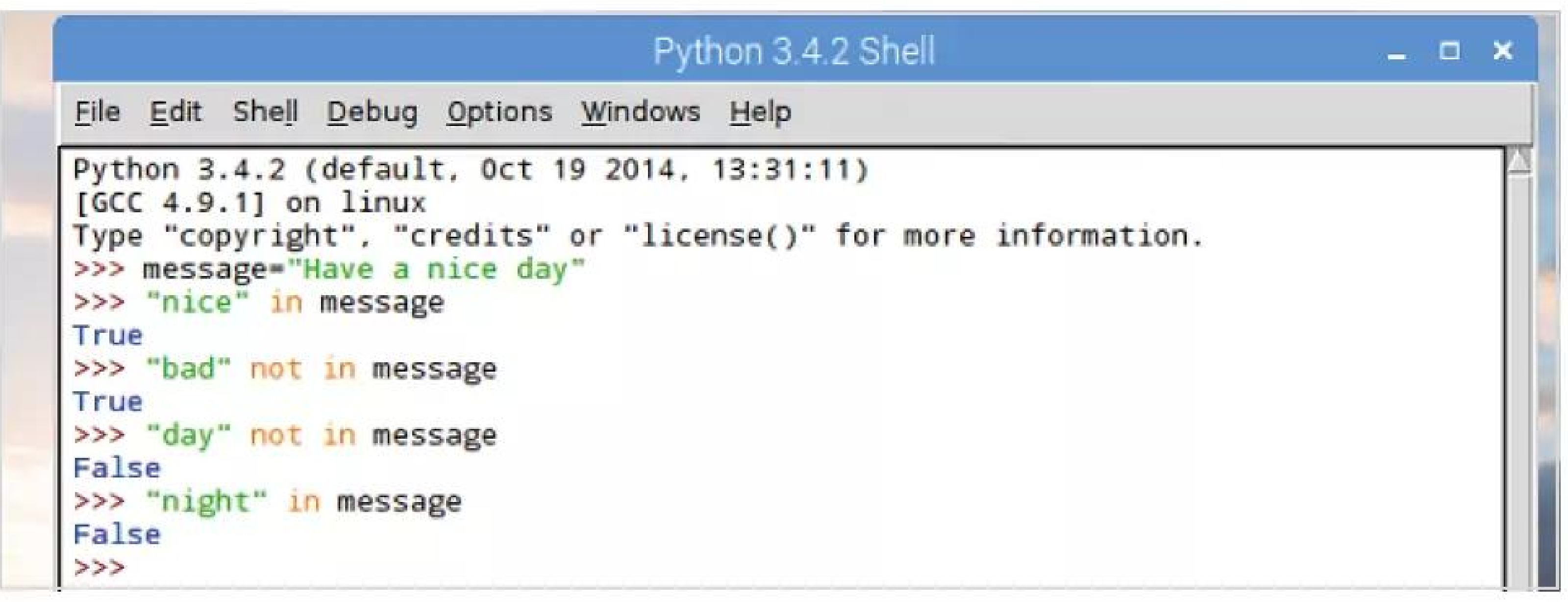

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> title="conan the cimmerian"
>>> title.capitalize()
'Conan the cimmerian'
>>> title.title()
'Conan The Cimmerian'
>>> |
```

STEP 10

You can also use logic operators on strings, with the 'in' and 'not in' functions. These enable you to check if a string contains (or does not contain) a sequence of characters:

```
message=”Have a nice day”
“nice” in message

”bad” not in message
“day” not in message
“night” in message
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> message="Have a nice day"
>>> "nice" in message
True
>>> "bad" not in message
True
>>> "day" not in message
False
>>> "night" in message
False
>>>
```

Formatting Strings

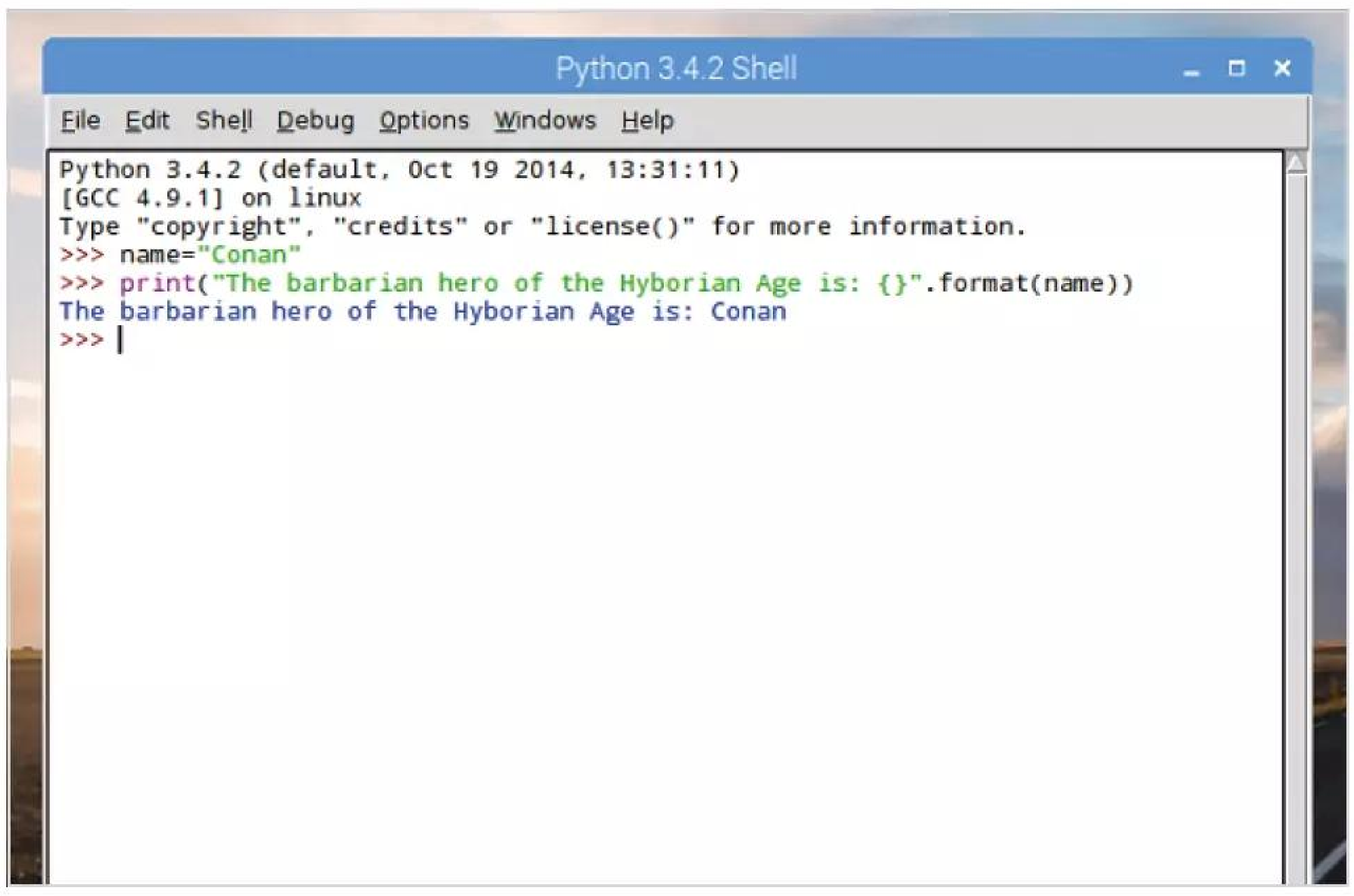
When you work with data, creating lists, dictionaries and objects you may often want to print out the results. Merging strings with data is easy especially with Python 3, as earlier versions of Python tended to complicate matters.

STRING FORMATTING

Since Python 3, string formatting has become a much neater process, using the .format function combined with curly brackets. It's a more logical and better formed approach than previous versions.

STEP 1 The basic formatting in Python is to call each variable into the string using the curly brackets:

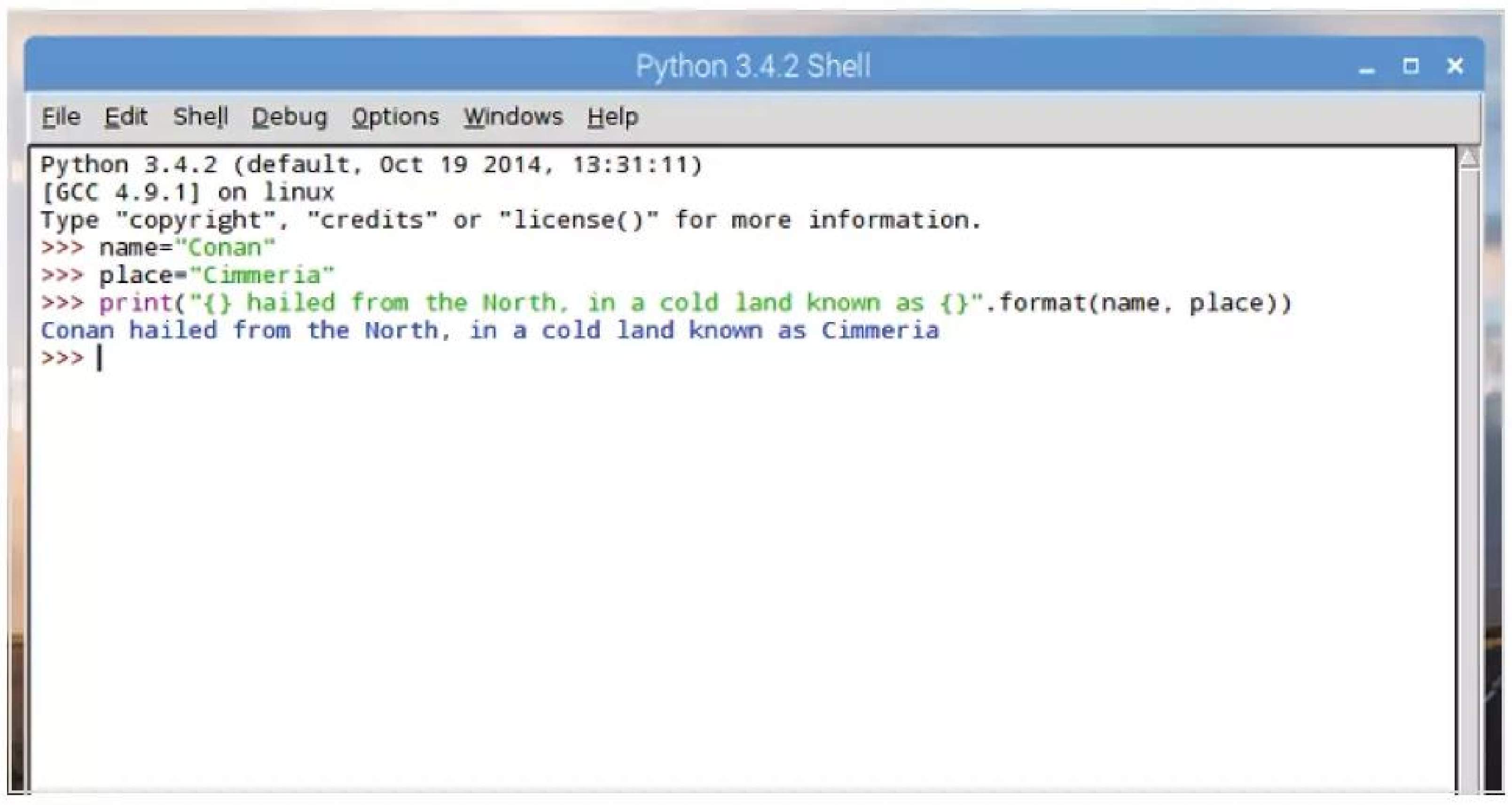
```
name=”Conan”
print(“The barbarian hero of the Hyborian Age is:
{}”.format(name))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="Conan"
>>> print("The barbarian hero of the Hyborian Age is: {}".format(name))
The barbarian hero of the Hyborian Age is: Conan
>>> |
```

STEP 2 Remember to close the print function with two sets of brackets, as you've encased the variable in one, and the print function in another. You can include multiple cases of string formatting in a single print function:

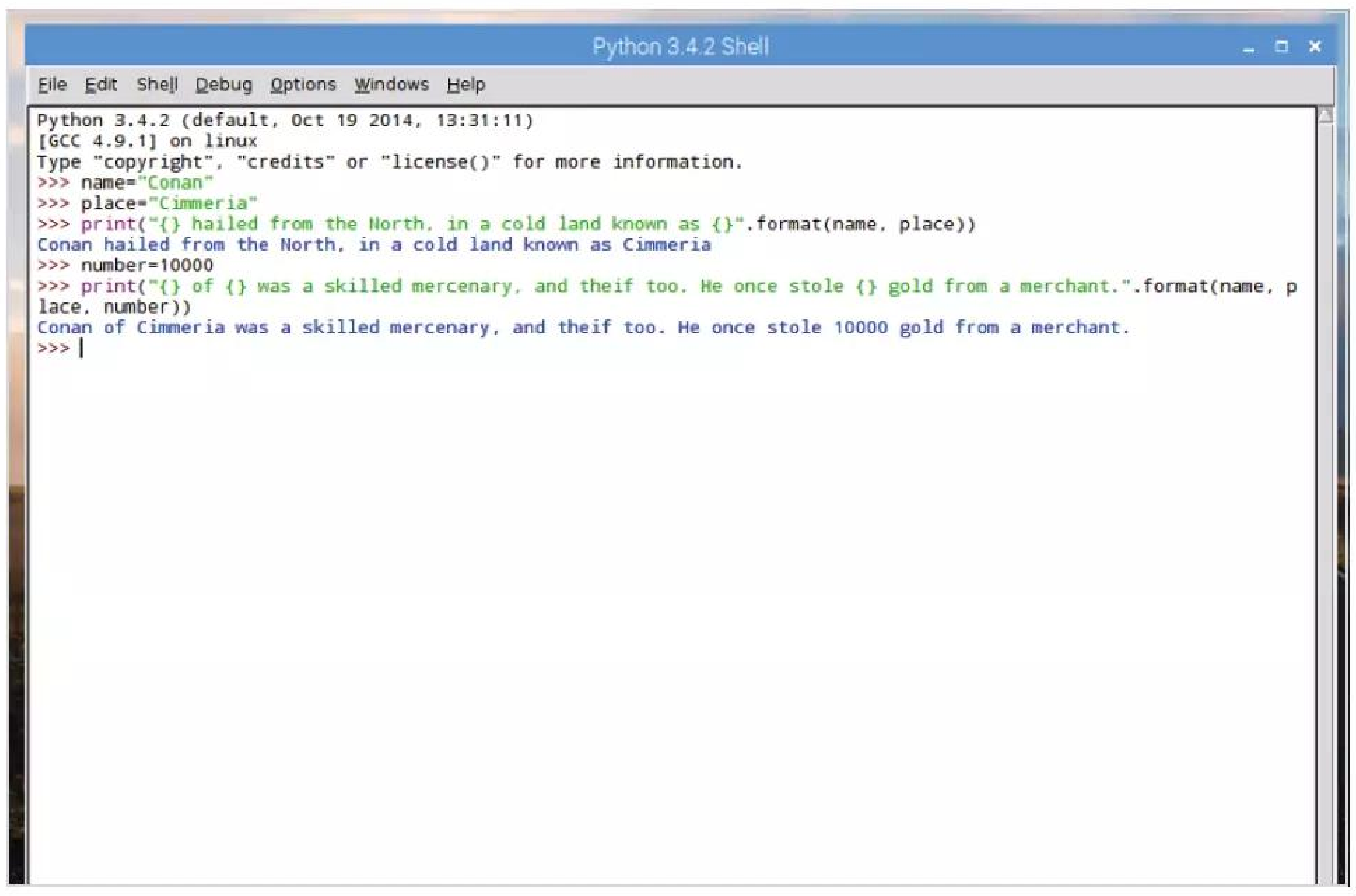
```
name=”Conan”
place=”Cimmeria”
print(“{} hailed from the North, in a cold land
known as {}”.format(name, place))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="Conan"
>>> place="Cimmeria"
>>> print("{} hailed from the North, in a cold land known as {}".format(name, place))
Conan hailed from the North, in a cold land known as Cimmeria
>>> |
```

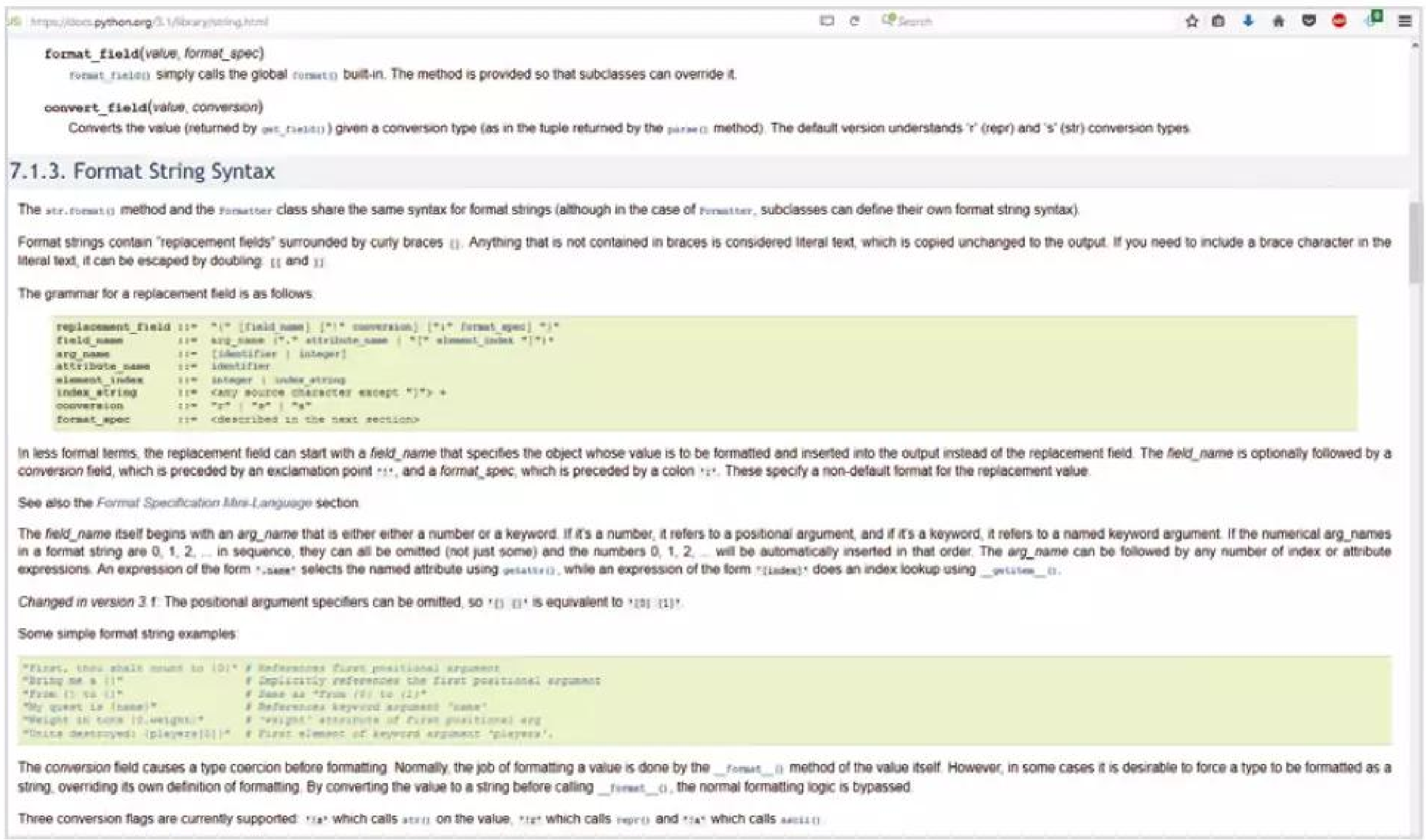
STEP 3 You can of course also include integers into the mix:

```
number=10000
print(“{} of {} was a skilled mercenary,
and thief too. He once stole {} gold from a
merchant.”.format(name, place, number))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="Conan"
>>> place="Cimmeria"
>>> print("{} hailed from the North, in a cold land known as {}".format(name, place))
Conan hailed from the North, in a cold land known as Cimmeria
>>> number=10000
>>> print("{} of {} was a skilled mercenary, and theif too. He once stole {} gold from a merchant.".format(name, p
lace, number))
Conan of Cimmeria was a skilled mercenary, and theif too. He once stole 10000 gold from a merchant.
>>> |
```

STEP 4 There are many different ways to apply string formatting, some are quite simple, as we've shown you here; others can be significantly more complex. It all depends on what you want from your program. A good place to reference frequently regarding string formatting is the Python Docs webpage, found at **www.docs.python.org/3.1/library/string.html**. Here, you will find tons of help.



format_field(*value, format_spec*)

`format_field()` simply calls the global `format()` built-in. The method is provided so that subclasses can override it.

convert_field(*value, conversion*)

Converts the value (returned by `get_field()`) given a conversion type (as in the tuple returned by the `parse()` method). The default version understands 'r' (repr) and 's' (str) conversion types.

7.1.3. Format String Syntax

The `str.format()` method and the `Formatter` class share the same syntax for format strings (although in the case of `Formatter`, subclasses can define their own format string syntax).

Format strings contain "replacement fields" surrounded by curly braces `{}`. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output. If you need to include a brace character in the literal text, it can be escaped by doubling: `{{` and `}}`.

The grammar for a replacement field is as follows:

In less formal terms, the replacement field can start with a *field_name* that specifies the object whose value is to be formatted and inserted into the output instead of the replacement field. The *field_name* is optionally followed by a *conversion* field, which is preceded by an exclamation point `'!'`, and a *format_spec*, which is preceded by a colon `':'`. These specify a non-default format for the replacement value.

See also the *Format Specification Mini-Language* section.

The *field_name* itself begins with an *arg_name* that is either either a number or a keyword. If it's a number, it refers to a positional argument, and if it's a keyword, it refers to a named keyword argument. If the numerical arg_names in a format string are 0, 1, 2, ... in sequence, they can all be omitted (not just some) and the numbers 0, 1, 2, ... will be automatically inserted in that order. The *arg_name* can be followed by any number of index or attribute expressions. An expression of the form `'.name'` selects the named attribute using `getattr()`, while an expression of the form `'[index]'` does an index lookup using `__getitem__()`.

Changed in version 3.1: The positional argument specifiers can be omitted, so `'{} {}'` is equivalent to `'{0} {1}'`.

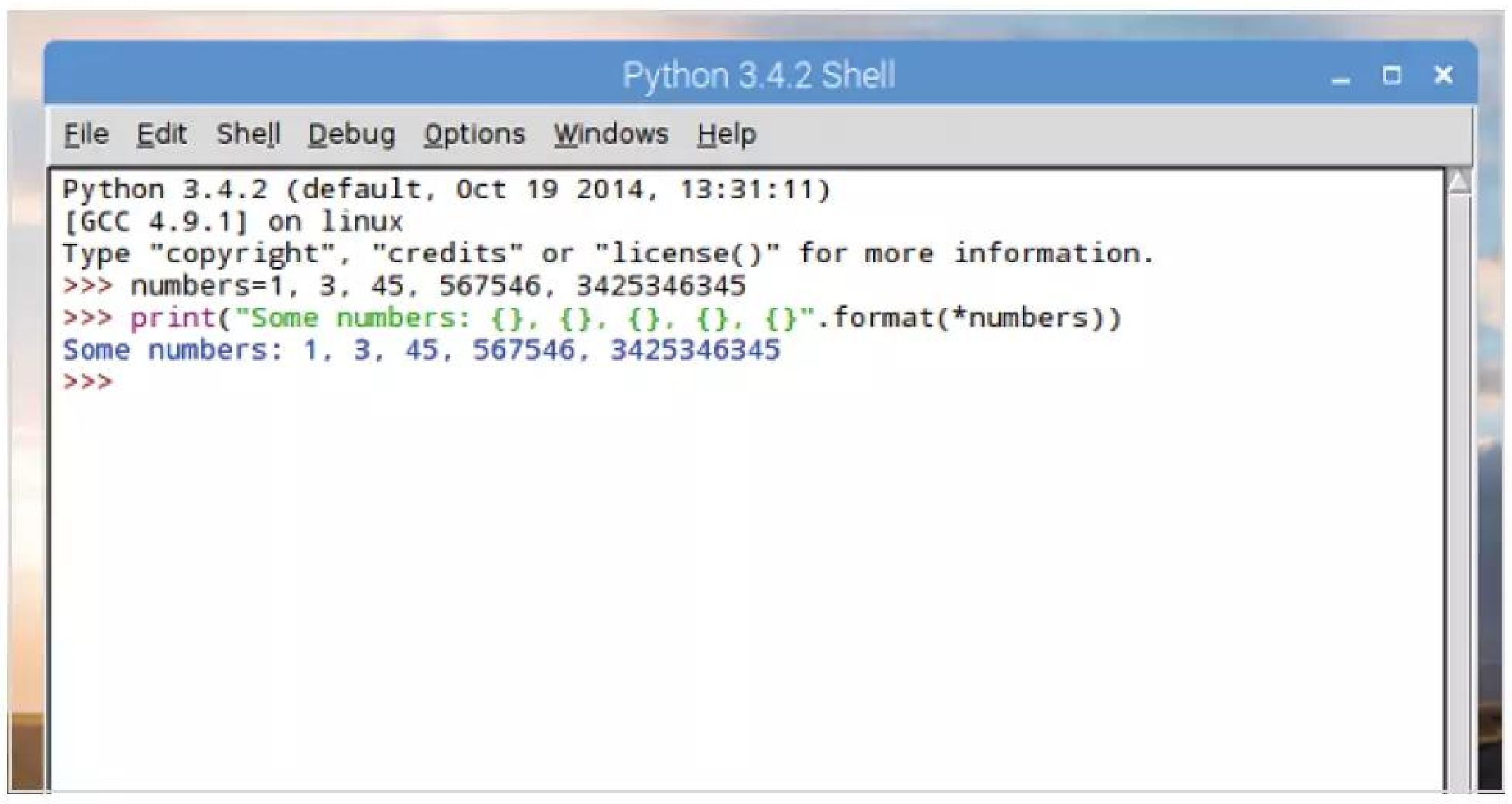
Some simple format string examples:

The *conversion* field causes a type coercion before formatting. Normally, the job of formatting a value is done by the `__format__()` method of the value itself. However, in some cases it is desirable to force a type to be formatted as a string, overriding its own definition of formatting. By converting the value to a string before calling `__format__()`, the normal formatting logic is bypassed.

Three conversion flags are currently supported: `'!s'` which calls `str()` on the value, `'!r'` which calls `repr()` and `'!a'` which calls `ascii()`.

STEP 5 Interestingly you can reference a list using the string formatting function. You need to place an asterisk in front of the list name:

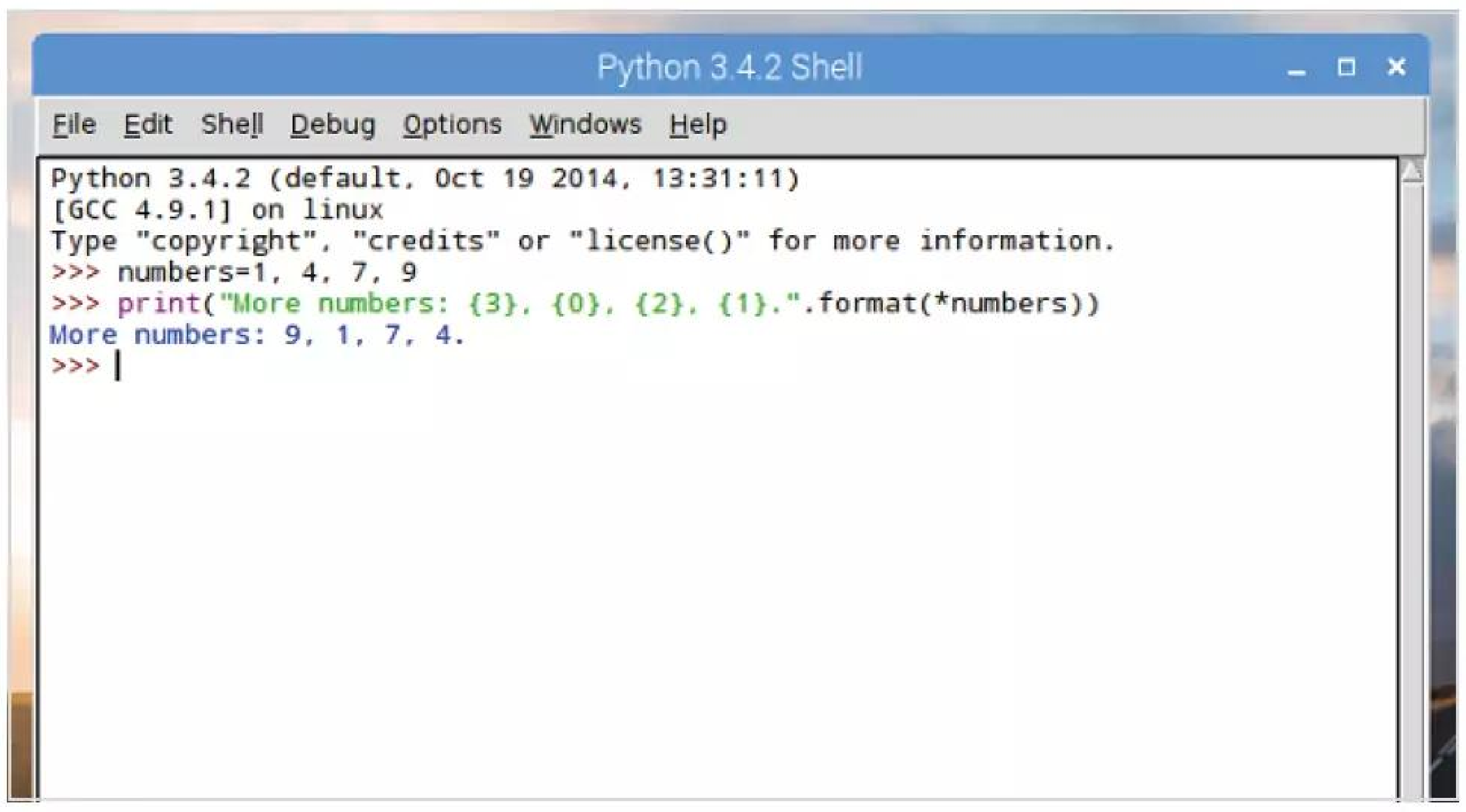
```
numbers=1, 3, 45, 567546, 3425346345
print("Some numbers: {}, {}, {}, {}, {}".
format(*numbers))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers=1, 3, 45, 567546, 3425346345
>>> print("Some numbers: {}, {}, {}, {}, {}".format(*numbers))
Some numbers: 1, 3, 45, 567546, 3425346345
>>>
```

STEP 6 With indexing in lists, the same applies to calling a list using string formatting. You can index each item according to its position (from 0 to however many are present):

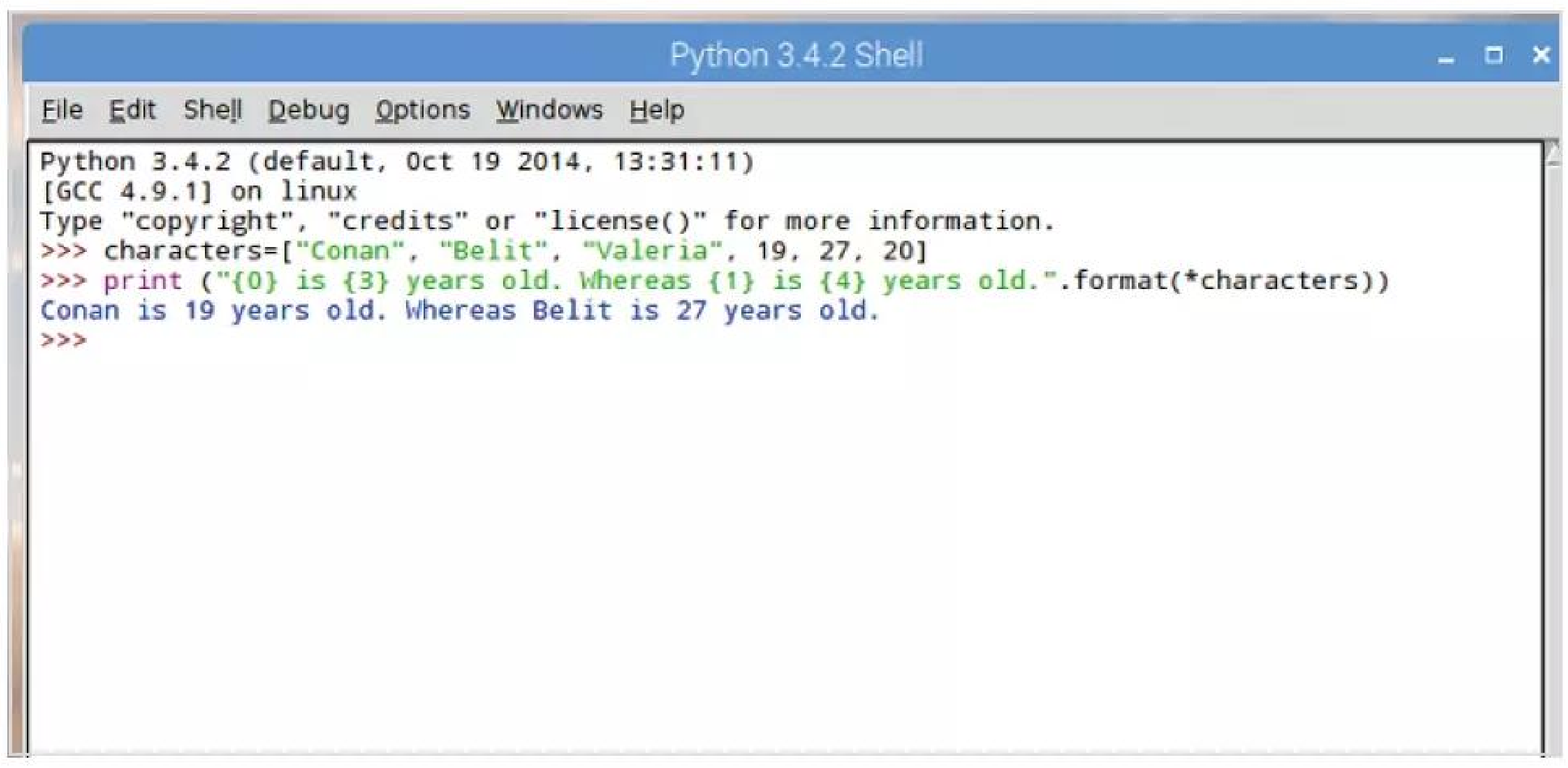
```
numbers=1, 4, 7, 9
print("More numbers: {3}, {0}, {2},
{1}.".format(*numbers))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> numbers=1, 4, 7, 9
>>> print("More numbers: {3}, {0}, {2}, {1}.".format(*numbers))
More numbers: 9, 1, 7, 4.
>>>
```

STEP 7 And as you probably suspect, you can mix strings and integers in a single list to be called in the .format function:

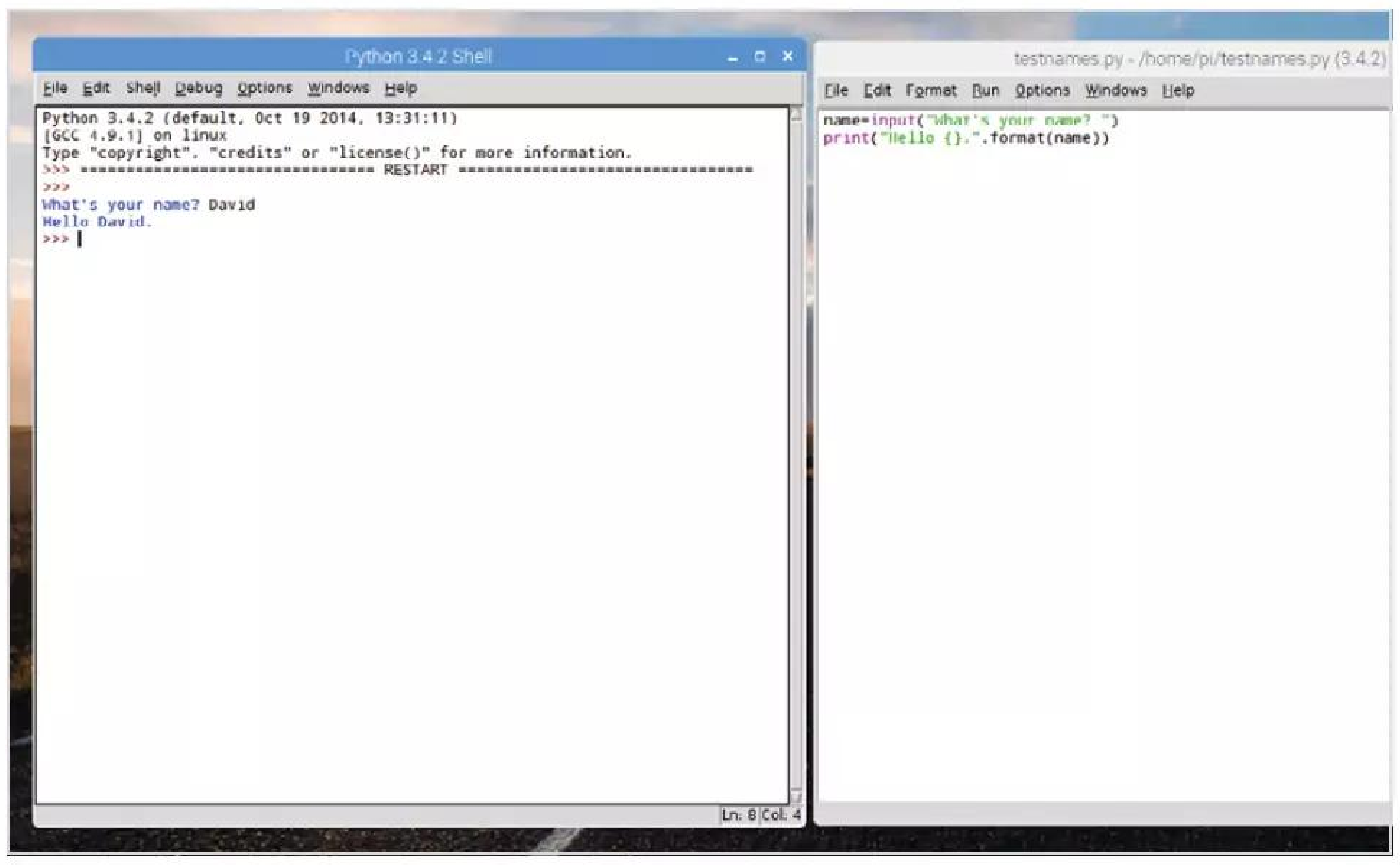
```
characters=["Conan", "Belit", "Valeria", 19, 27,
20]
print ("{0} is {3} years old. Whereas {1} is {4}
years old.".format(*characters))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> characters=["Conan", "Belit", "Valeria", 19, 27, 20]
>>> print ("{0} is {3} years old. Whereas {1} is {4} years old.".format(*characters))
Conan is 19 years old. Whereas Belit is 27 years old.
>>>
```

STEP 8 You can also print out the content of a user's input in the same fashion:

```
name=input("What's your name? ")
print("Hello {}.".format(name)
```

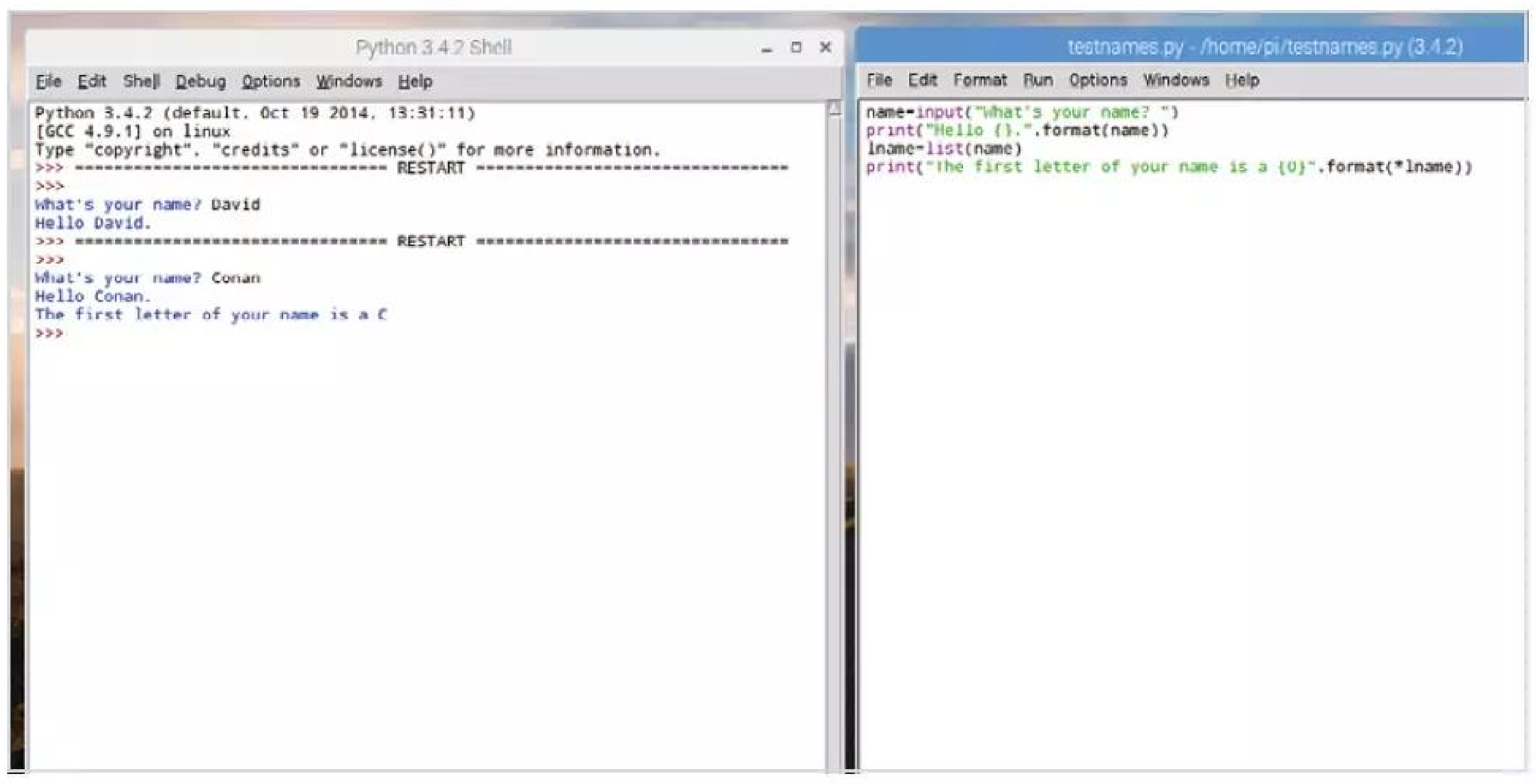

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
testnames.py - /home/pi/testnames.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 8 Col: 4

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What's your name? David
Hello David.
>>>
```

```
name=input("What's your name? ")
print("Hello {}.".format(name))
```

STEP 9 You can extend this simple code example to display the first letter in a person's entered name:

```
name=input("What's your name? ")
print("Hello {}.".format(name))
lname=list(name)
print("The first letter of your name is a {0}".
format(*lname))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
testnames.py - /home/pi/testnames.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
What's your name? David
Hello David.
>>> ================================ RESTART ================================
>>>
What's your name? Conan
Hello Conan.
The first letter of your name is a C
>>>
```

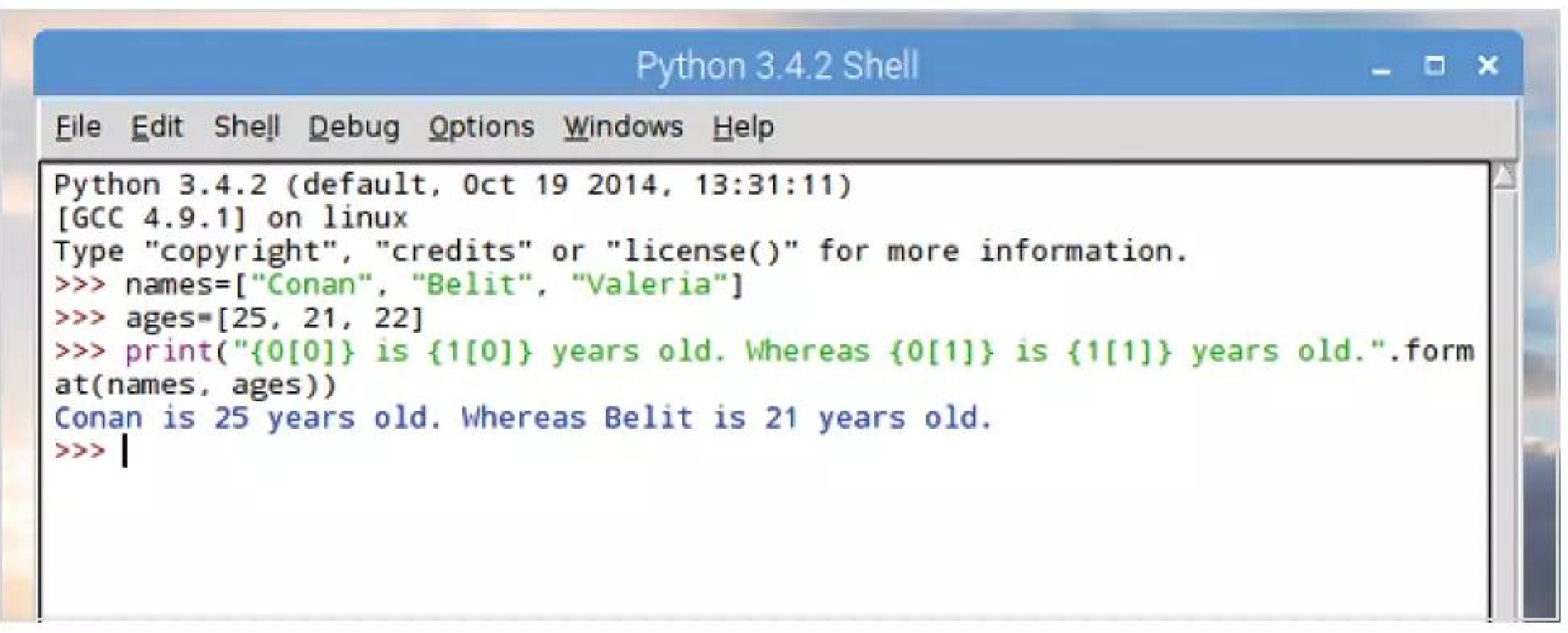
```
name=input("What's your name? ")
print("Hello {}.".format(name))
lname=list(name)
print("The first letter of your name is a {0}".format(*lname))
```

STEP 10 You can also call upon a pair of lists and reference them individually within the same print function. Looking back the code from Step 7, you can alter it with:

```
names=["Conan", "Belit", "Valeria"]
ages=[25, 21, 22]
```

Creating two lists. Now you can call each list, and individual items:

```
print("{0[0]} is {1[0]} years old. Whereas {0[1]}
is {1[1]} years old.".format(names, ages))
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> names=["Conan", "Belit", "Valeria"]
>>> ages=[25, 21, 22]
>>> print("{0[0]} is {1[0]} years old. Whereas {0[1]} is {1[1]} years old.".form
at(names, ages))
Conan is 25 years old. Whereas Belit is 21 years old.
>>>
```

Date and Time

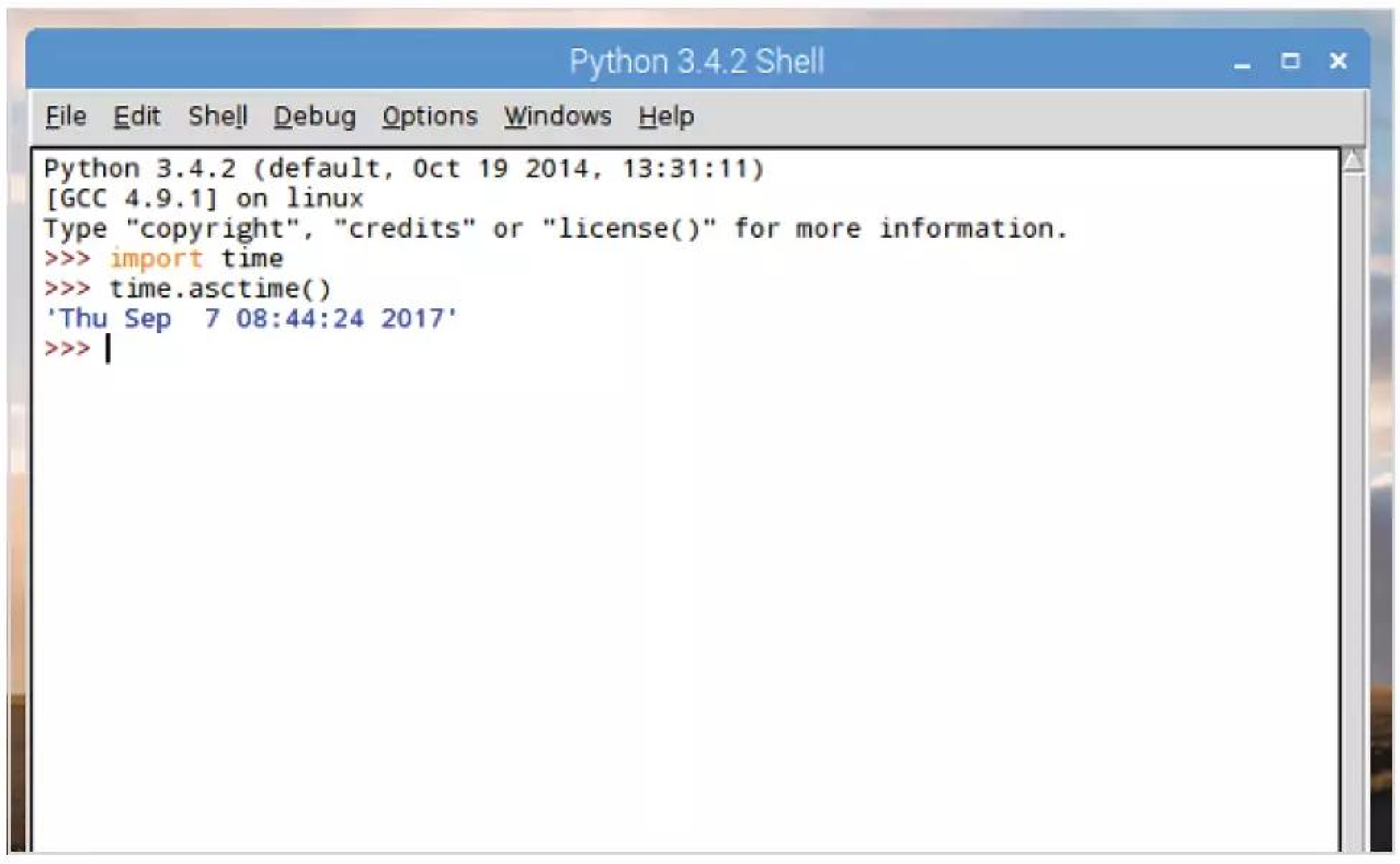
When working with data it's often handy to have access to the time. For example, you may want to time-stamp an entry or see at what time a user logged into the system and for how long. Luckily acquiring the date and time is easy, thanks to the Time module.

TIME LORDS

The time module contains functions that help you retrieve the current system time, reads the date from strings, formats the time and date and much more.

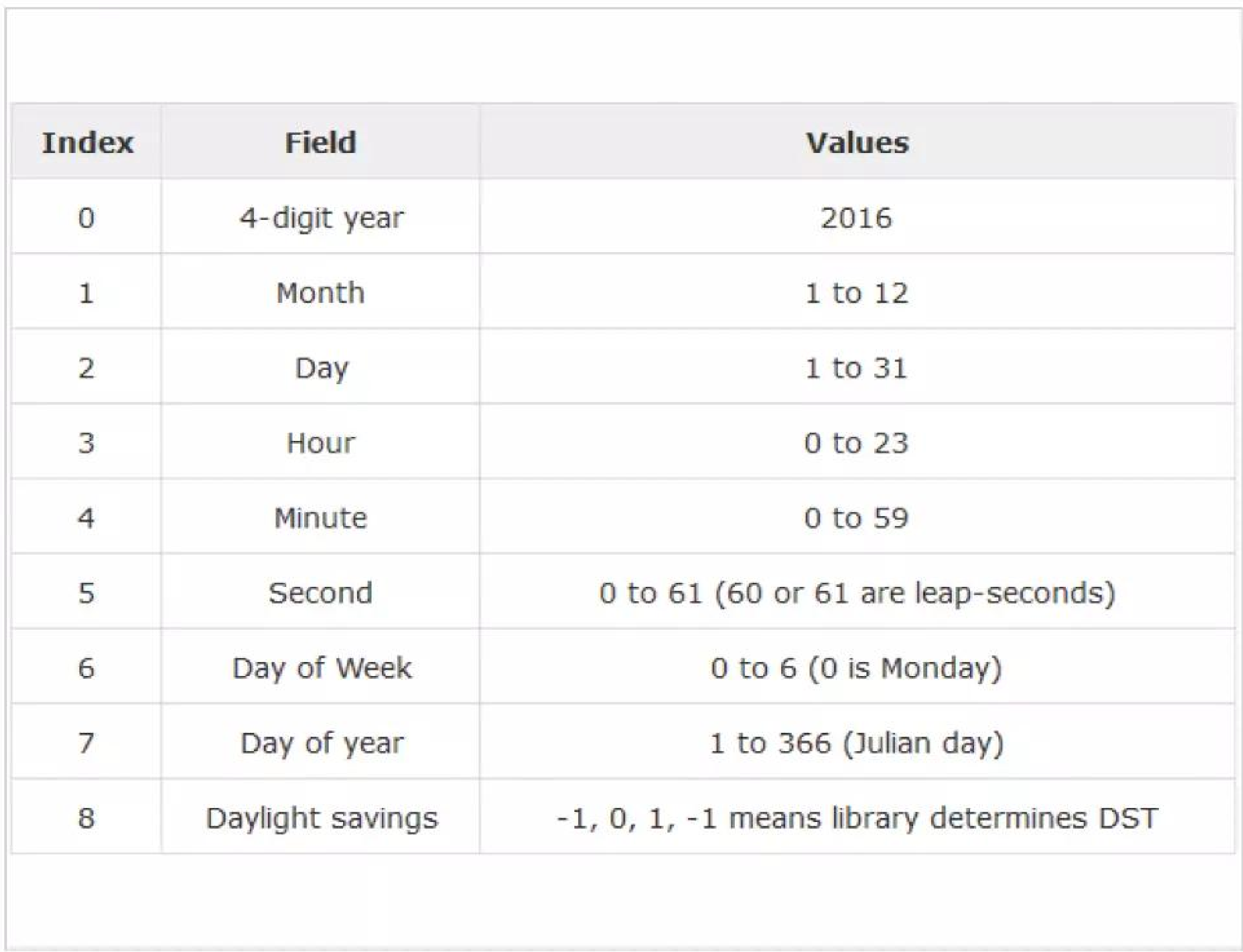
STEP 1 First you need to import the time module. It's one that's built-in to Python 3 so you shouldn't need to drop into a command prompt and pip install it. Once it's imported, you can call the current time and date with a simple command:

```
import time
time.asctime()
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.asctime()
'Thu Sep  7 08:44:24 2017'
>>> |
```

STEP 2 The time function is split into nine tuples, these are divided up into indexed items, as with any other tuple, and shown in the screen shot below.

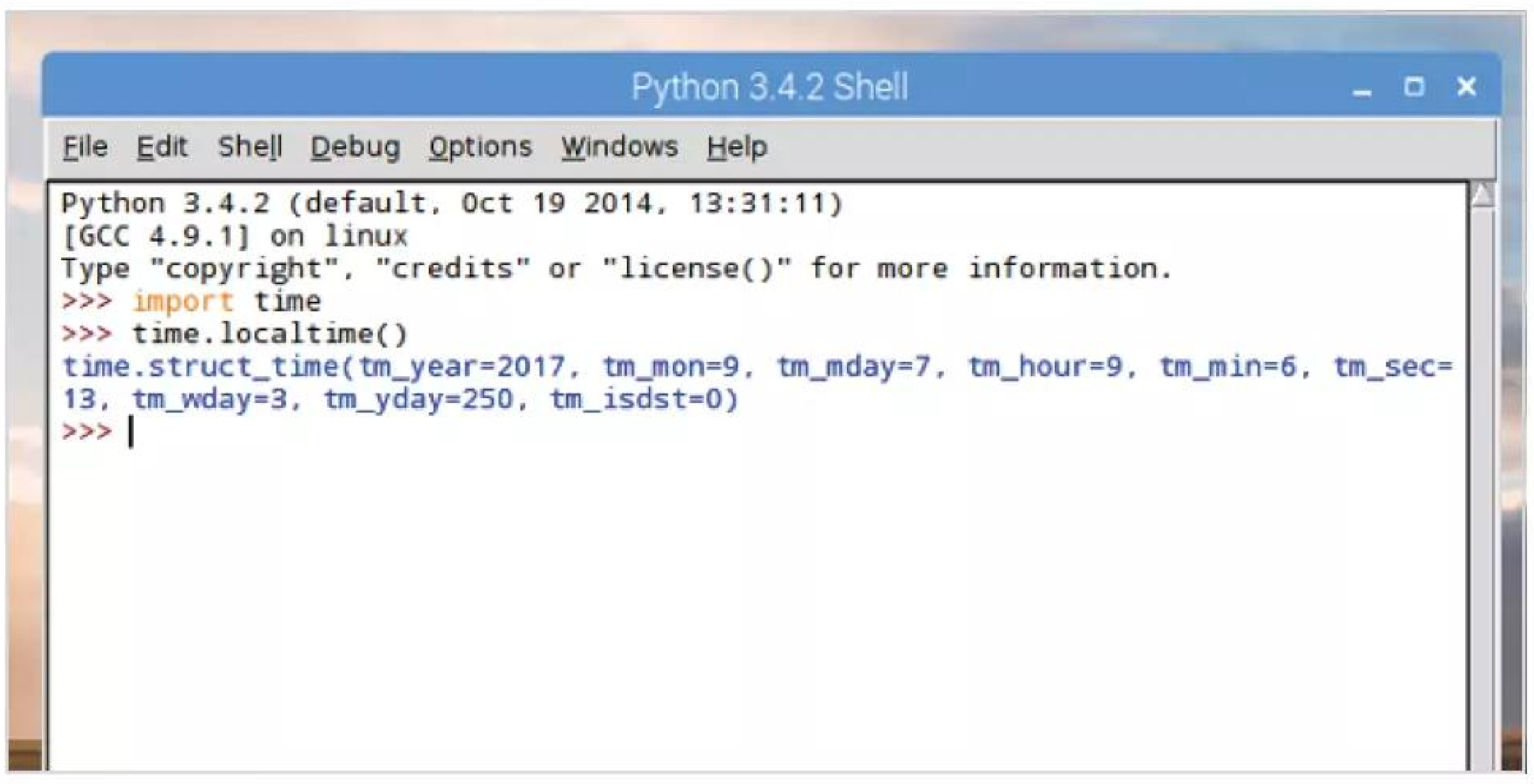


Index	Field	Values
0	4-digit year	2016
1	Month	1 to 12
2	Day	1 to 31
3	Hour	0 to 23
4	Minute	0 to 59
5	Second	0 to 61 (60 or 61 are leap-seconds)
6	Day of Week	0 to 6 (0 is Monday)
7	Day of year	1 to 366 (Julian day)
8	Daylight savings	-1, 0, 1, -1 means library determines DST

STEP 3 You can see the structure of how time is presented by entering:

```
time.local.time()
```

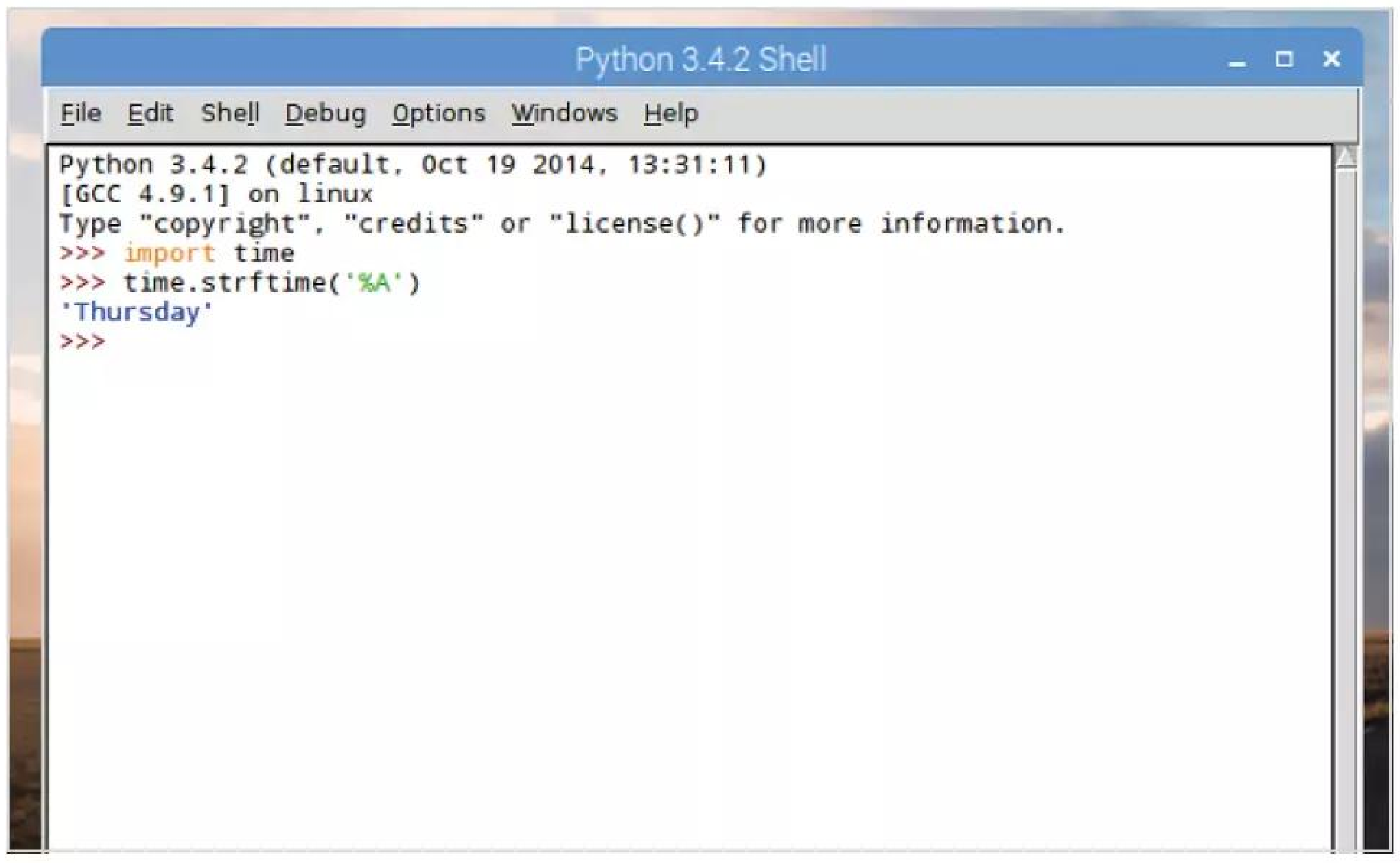
The output is displayed as such: `'time.struct_time(tm_year=2017, tm_mon=9, tm_mday=7, tm_hour=9, tm_min=6, tm_sec=13, tm_wday=3, tm_yday=250, tm_isdst=0)'`; obviously dependent on your current time as opposed to the time shown above.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.localtime()
time.struct_time(tm_year=2017, tm_mon=9, tm_mday=7, tm_hour=9, tm_min=6, tm_sec=
13, tm_wday=3, tm_yday=250, tm_isdst=0)
>>> |
```

STEP 4 There are numerous functions built into the time module. One of the most common of these is .strftime(). With it, you're able to present a wide range of arguments as it converts the time tuple into a string. For example, to display the current day of the week you can use:

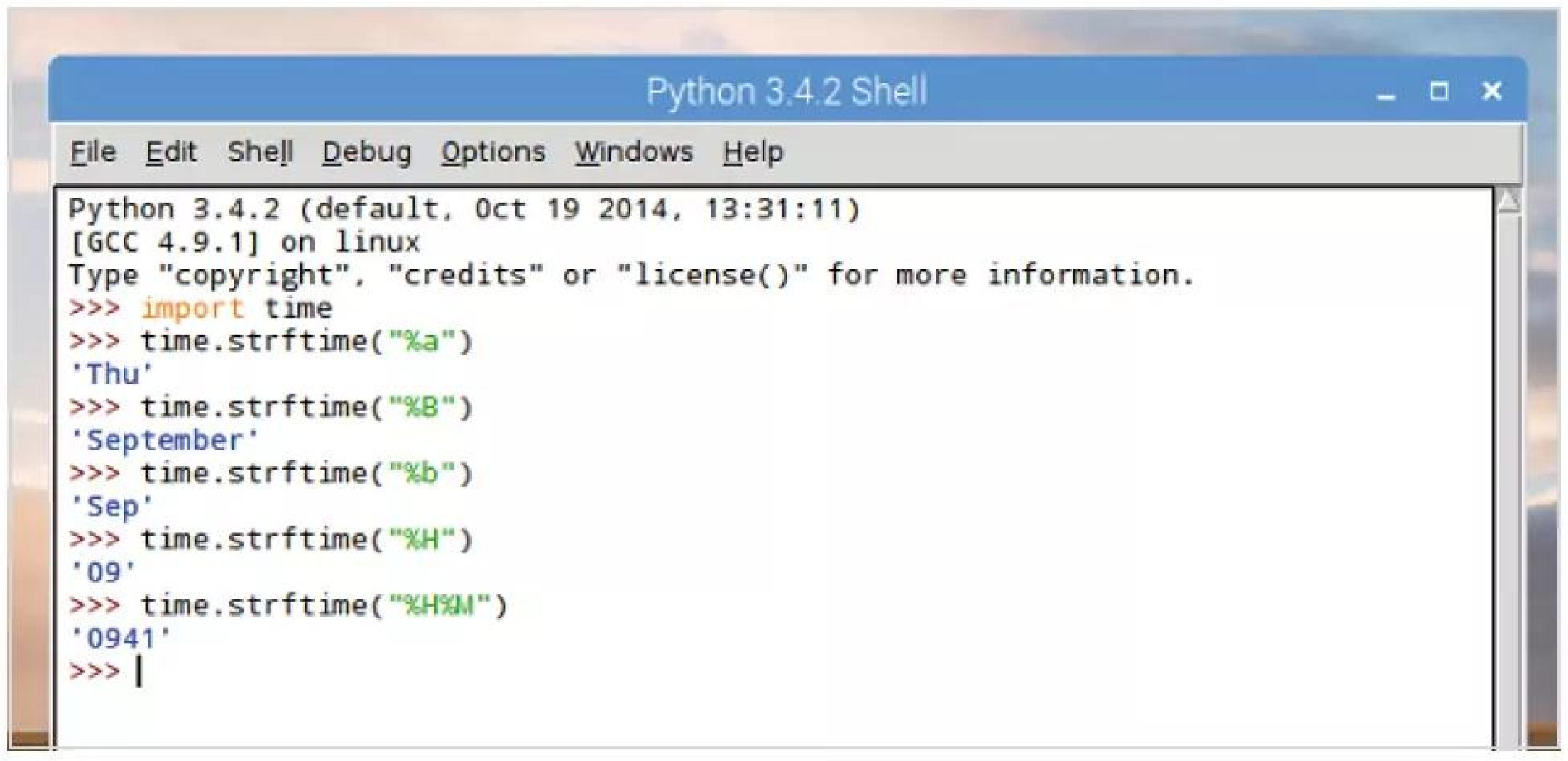
```
time.strftime('%A')
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime('%A')
'Thursday'
>>>
```

STEP 5 This naturally means you can incorporate various functions into your own code, such as:

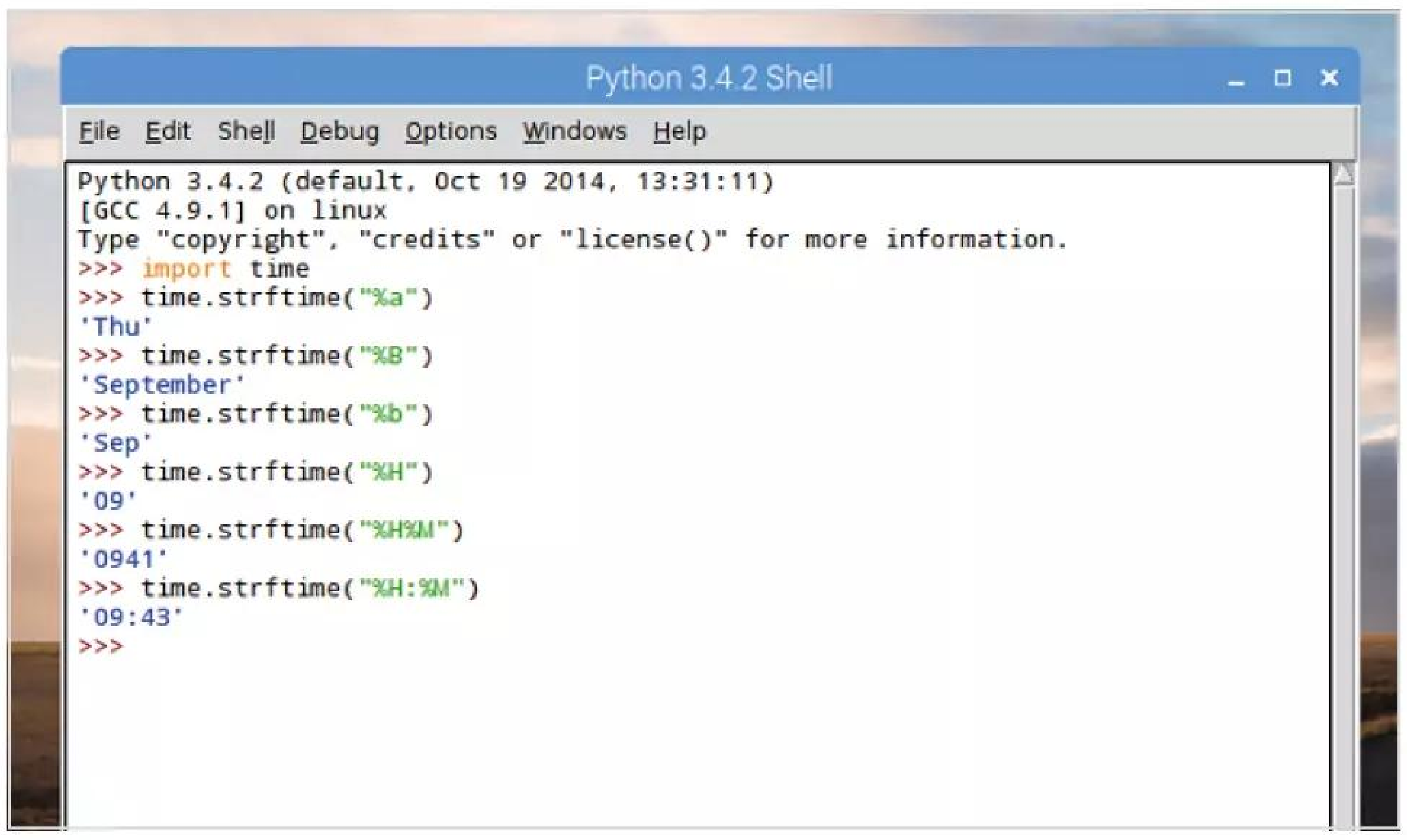
```
time.strftime("%a")
time.strftime("%B")
time.strftime("%b")
time.strftime("%H")
time.strftime("%H%M")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime("%a")
'Thu'
>>> time.strftime("%B")
'September'
>>> time.strftime("%b")
'Sep'
>>> time.strftime("%H")
'09'
>>> time.strftime("%H%M")
'0941'
>>> |
```

STEP 6 Note the last two entries, with %H and %H%M, as you can see these are the hours and minutes and as the last entry indicates, entering them as %H%M doesn't display the time correctly in the Shell. You can easily rectify this with:

```
time.strftime("%H:%M")
```

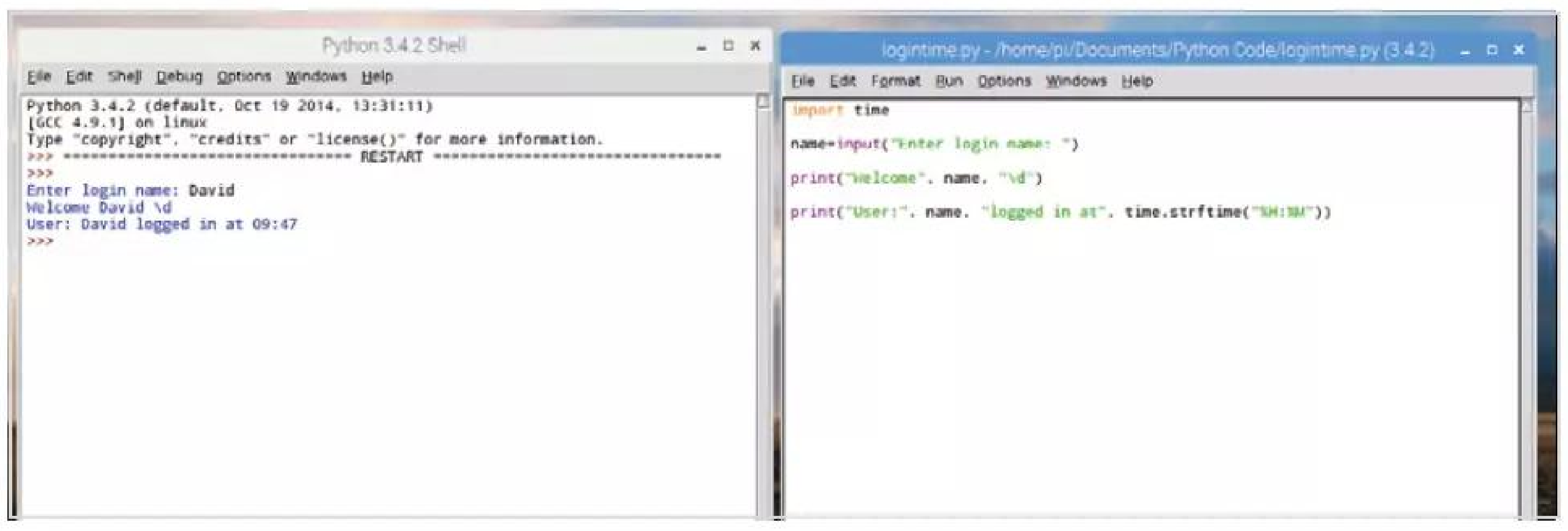

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> time.strftime("%a")
'Thu'
>>> time.strftime("%B")
'September'
>>> time.strftime("%b")
'Sep'
>>> time.strftime("%H")
'09'
>>> time.strftime("%H%M")
'0941'
>>> time.strftime("%H:%M")
'09:43'
>>>
```

STEP 7 This means you're going to be able to display either the current time or the time when something occurred, such as a user entering their name. Try this code in the Editor:

```
import time
name=input("Enter login name: ")
print("Welcome", name, "\d")
print("User:", name, "logged in at", time.
strftime("%H:%M"))
```

Try to extend it further to include day, month, year and so on.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
logintime.py - /home/pi/Documents/Python Code/logintime.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter login name: David
Welcome David \d
User: David logged in at 09:47
>>>
```

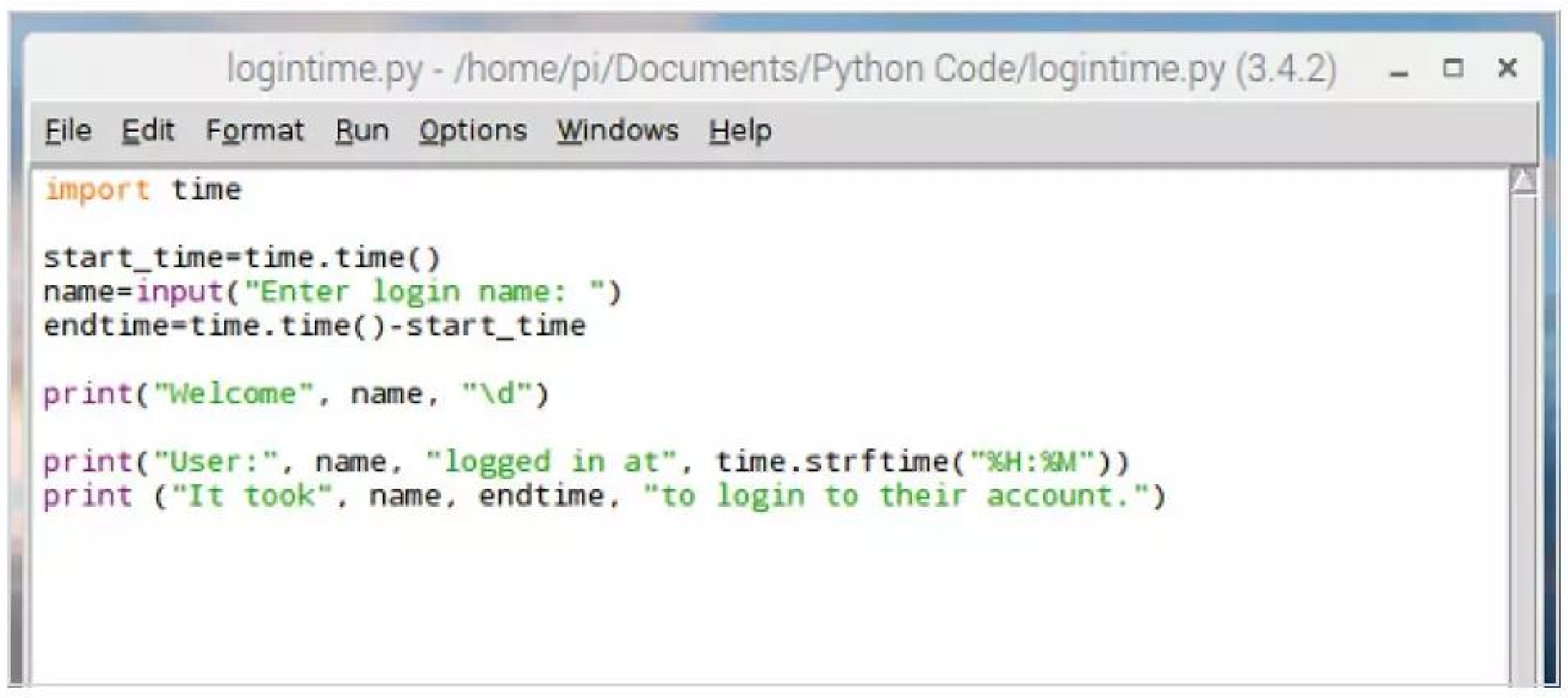
```
import time

name=input("Enter login name: ")

print("Welcome", name, "\d")

print("User:", name, "logged in at", time.strftime("%H:%M"))
```

STEP 8 You saw at the end of the previous section, in the code to calculate Pi to however many decimal places the users wanted, you can time a particular event in Python. Take the code from above and alter it slightly by including:

```
start_time=time.time()
```

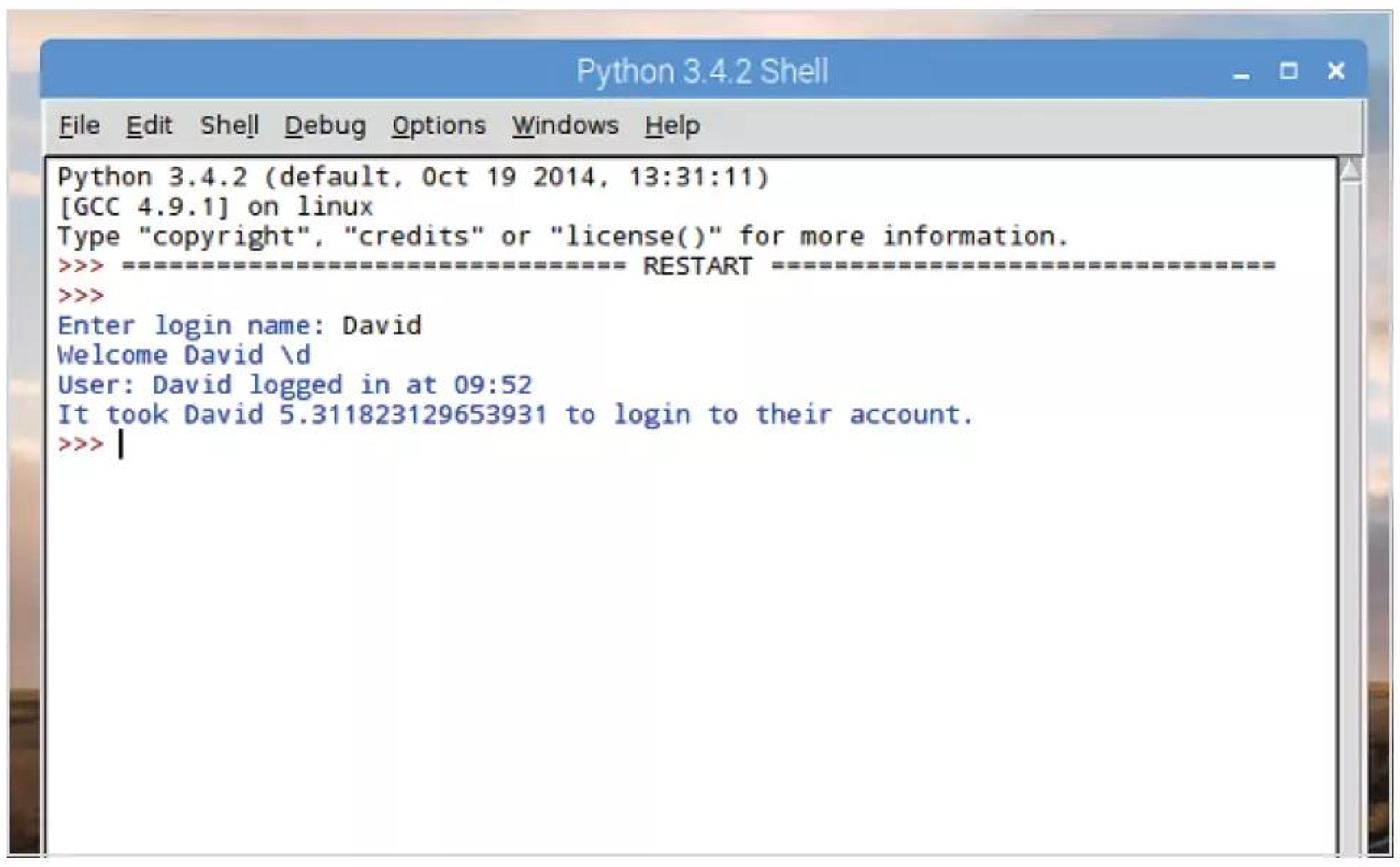
Then there's:

```
endtime=time.time()-start_time
```


logintime.py - /home/pi/Documents/Python Code/logintime.py (3.4.2)
File Edit Format Run Options Windows Help

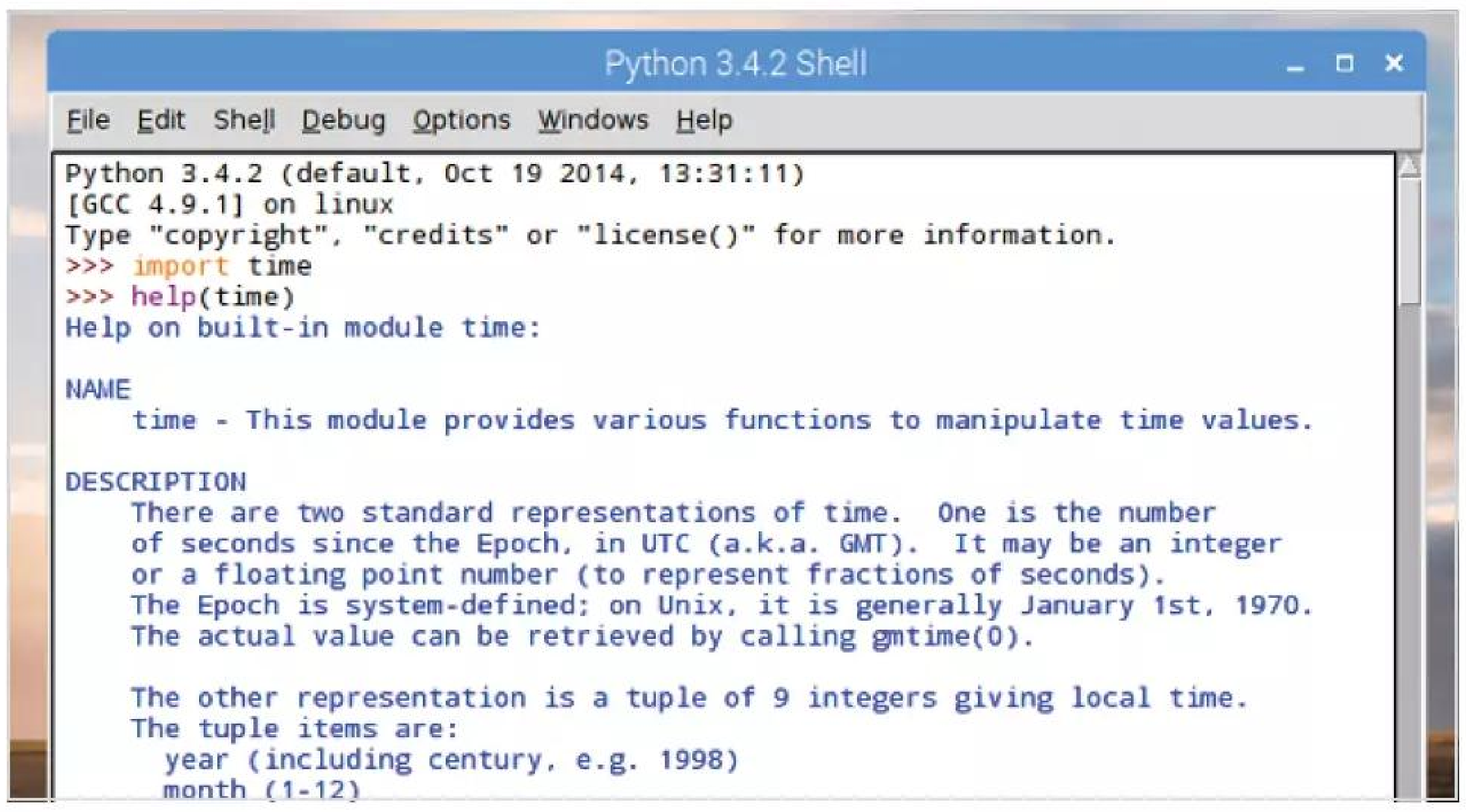
```
import time

start_time=time.time()
name=input("Enter login name: ")
endtime=time.time()-start_time

print("Welcome", name, "\d")

print("User:", name, "logged in at", time.strftime("%H:%M"))
print ("It took", name, endtime, "to login to their account.")
```

STEP 9 The output will look similar to the screenshot below. The timer function needs to be either side of the input statement, as that's when the variable name is being created, depending on how long the user took to log in. The length of time is then displayed on the last line of the code as the `endtime` variable.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter login name: David
Welcome David \d
User: David logged in at 09:52
It took David 5.311823129653931 to login to their account.
>>> |
```

STEP 10 There's a lot that can be done with the time module; some of it is quite complex too, such as displaying the number of seconds since January 1st 1970. If you want to drill down further into the time module, then in the Shell enter: `help(time)` to display the current Python version help file for the time module.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import time
>>> help(time)
Help on built-in module time:

NAME
    time - This module provides various functions to manipulate time values.

DESCRIPTION
    There are two standard representations of time.  One is the number
    of seconds since the Epoch, in UTC (a.k.a. GMT).  It may be an integer
    or a floating point number (to represent fractions of seconds).
    The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
    The actual value can be retrieved by calling gmtime(0).

    The other representation is a tuple of 9 integers giving local time.
    The tuple items are:
      year (including century, e.g. 1998)
      month (1-12)
```

Opening Files

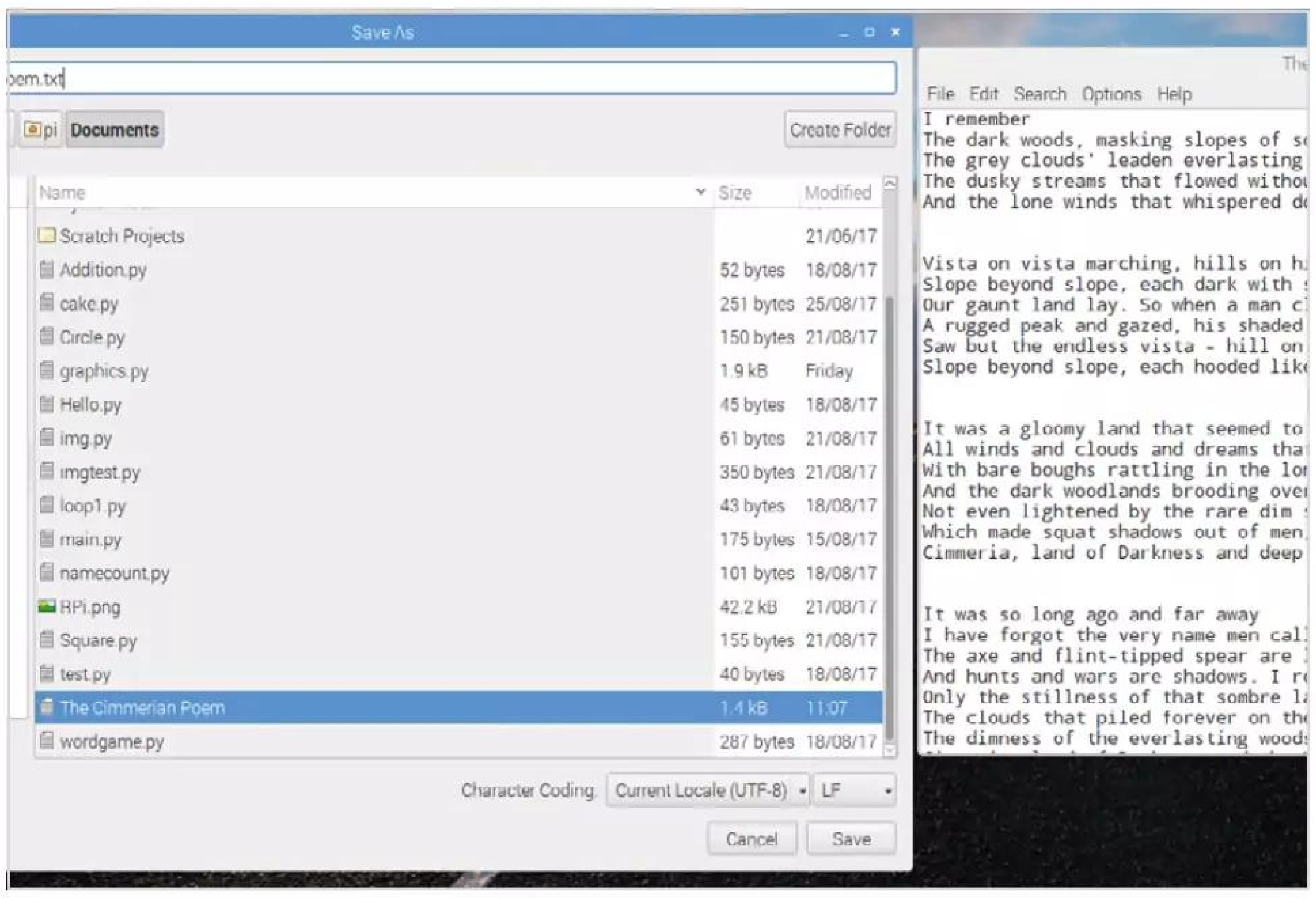
In Python you can read text and binary files in your programs. You can also write to file, which is something we will look at next. Reading and writing to files enables you to output and store data from your programs.

OPEN, READ AND WRITE

In Python you create a file object, similar to creating a variable, only pass in the file using the open() function. Files are usually categorised as text or binary.

STEP 1

Start by entering some text into your system's text editor. The text editor is best, not a word processor, as word processors include background formatting and other elements. In our example, we have the poem The Cimmerian, by Robert E Howard. You need to save the file as poem.txt.



oem.txt

Name	Size	Modified
Scratch Projects		21/06/17
Addition.py	52 bytes	18/08/17
cake.py	251 bytes	25/08/17
Circle.py	150 bytes	21/08/17
graphics.py	1.9 kB	Friday
Hello.py	45 bytes	18/08/17
img.py	61 bytes	21/08/17
imgtest.py	350 bytes	21/08/17
loop1.py	43 bytes	18/08/17
main.py	175 bytes	15/08/17
namecount.py	101 bytes	18/08/17
RPi.png	42.2 kB	21/08/17
Square.py	155 bytes	21/08/17
test.py	40 bytes	18/08/17
The Cimmerian Poem	1.4 kB	11:07
wordgame.py	287 bytes	18/08/17

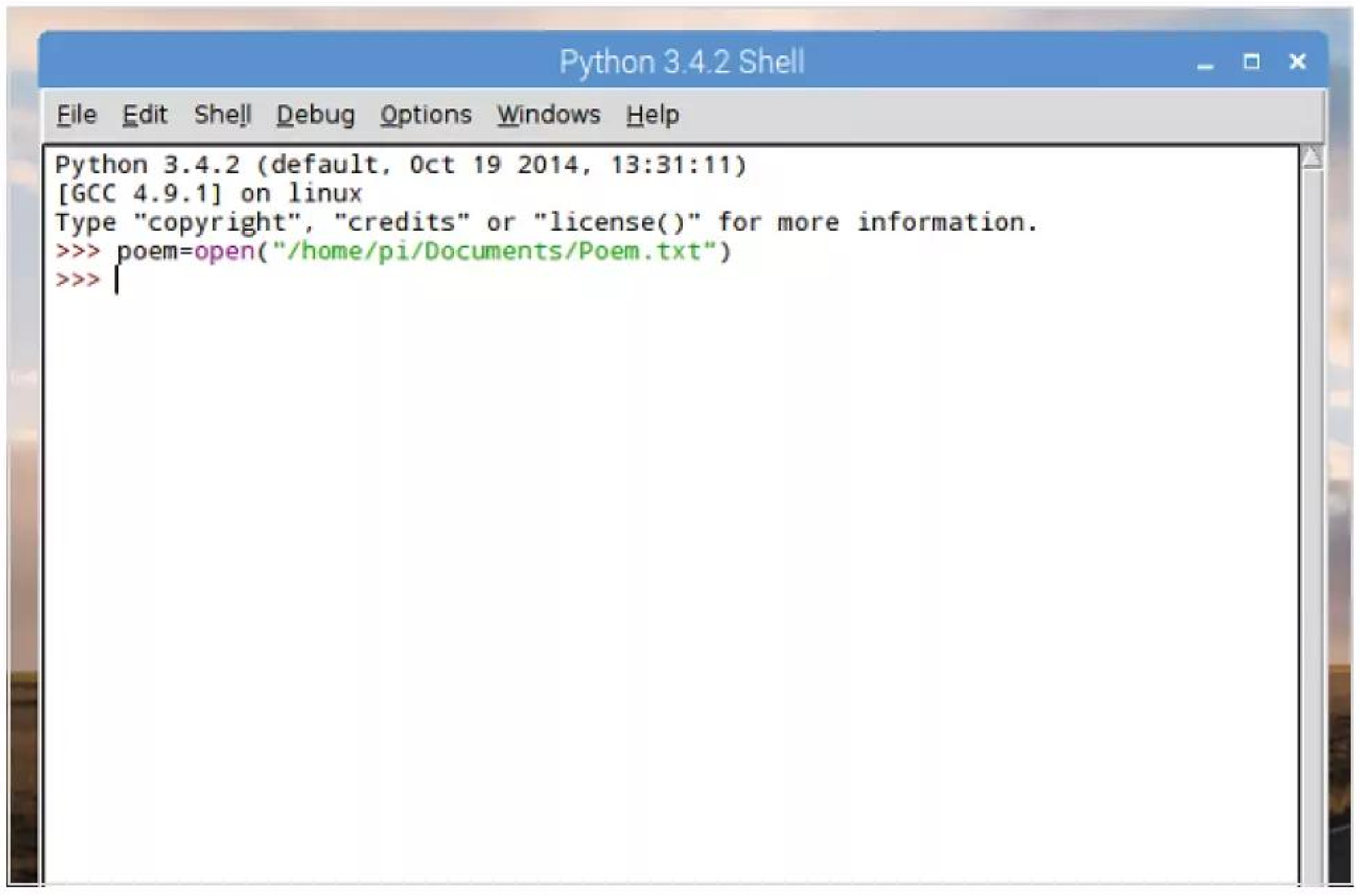
Character Coding: Current Locale (UTF-8) LF

```
I remember
The dark woods, masking slopes of so
The grey clouds' leaden everlasting
The dusky streams that flowed withou
And the lone winds that whispered do


Vista on vista marching, hills on hi
Slope beyond slope, each dark with s
Our gaunt land lay. So when a man cl
A rugged peak and gazed, his shaded
Saw but the endless vista - hill on
Slope beyond slope, each hooded like


It was a gloomy land that seemed to
All winds and clouds and dreams tha
With bare boughs rattling in the lon
And the dark woodlands brooding over
Not even lightened by the rare dim s
Which made squat shadows out of men,
Cimmeria, land of Darkness and deep


It was so long ago and far away
I have forgot the very name men cal
The axe and flint-tipped spear are l
And hunts and wars are shadows. I re
Only the stillness of that sombre la
The clouds that piled forever on the
The dimness of the everlasting wood
```

STEP 2

You use the open() function to pass the file into a variable as an object. You can name the file object anything you like, but you will need to tell Python the name and location of the text file you're opening:

```
poem=open("/home/pi/Documents/Poem.txt")
```



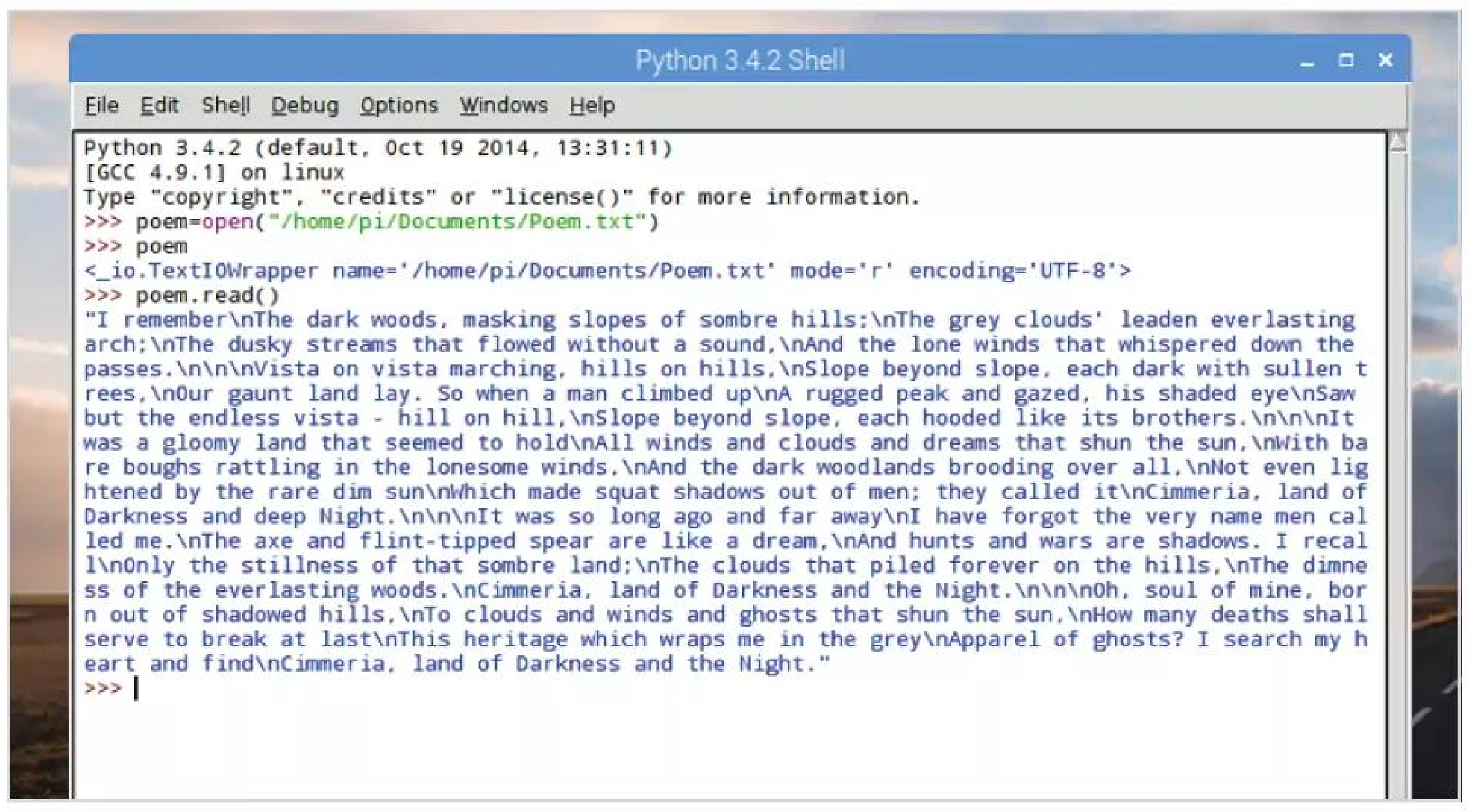
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> 
```

STEP 3

If you now enter poem into the Shell, you will get some information regarding the text file you've just asked to be opened. You can now use the poem variable to read the contents of the file:

```
poem.read()
```

Note than a /n entry in the text represents a new line, as you used previously.



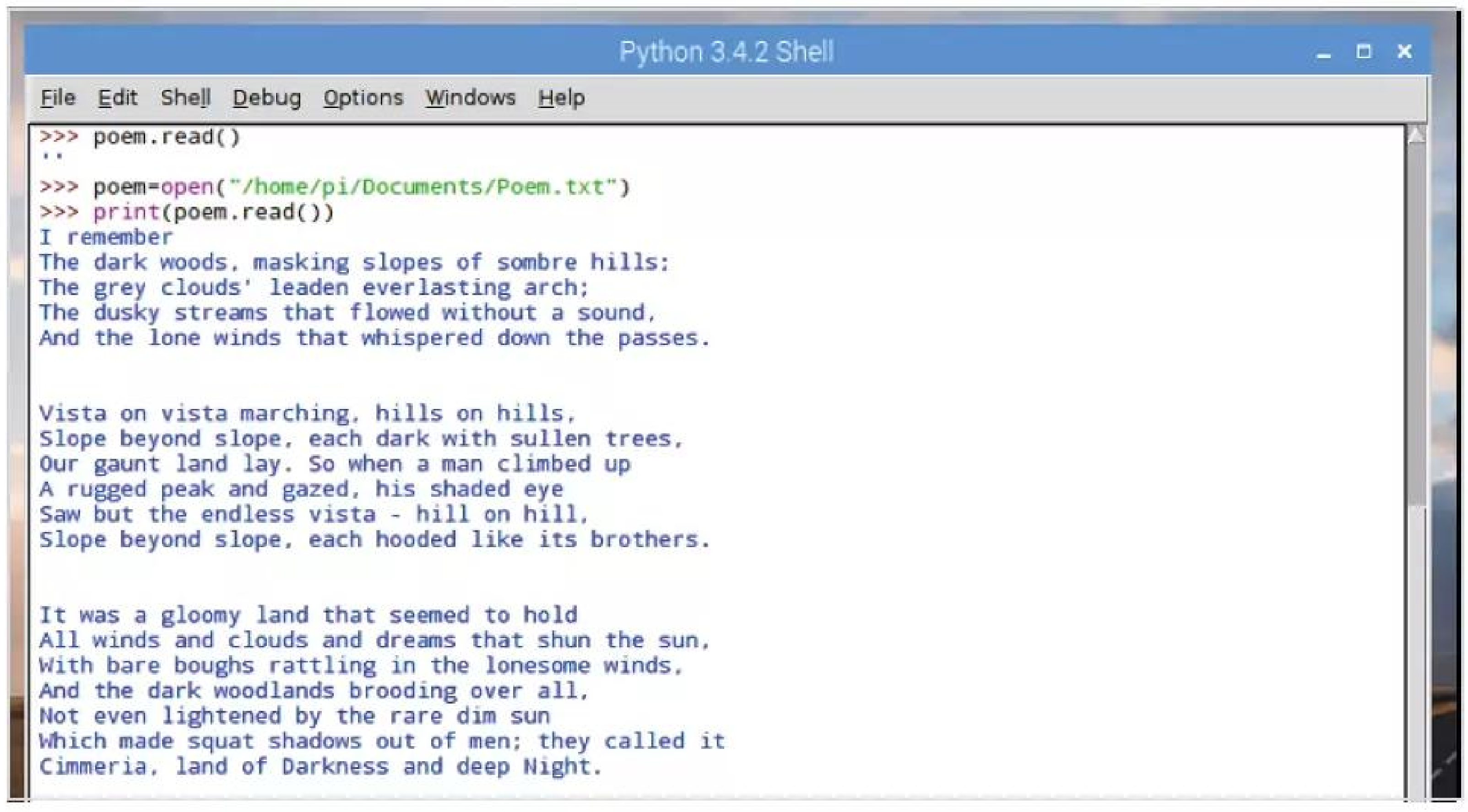
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem
<_io.TextIOWrapper name='/home/pi/Documents/Poem.txt' mode='r' encoding='UTF-8'>
>>> poem.read()
"I remember\nThe dark woods, masking slopes of sombre hills:\nThe grey clouds' leaden everlasting
arch;\nThe dusky streams that flowed without a sound,\nAnd the lone winds that whispered down the
passes.\n\n\nVista on vista marching, hills on hills,\nSlope beyond slope, each dark with sullen t
rees,\nOur gaunt land lay. So when a man climbed up\nA rugged peak and gazed, his shaded eye\nSaw
but the endless vista - hill on hill,\nSlope beyond slope, each hooded like its brothers.\n\n\nIt
was a gloomy land that seemed to hold\nAll winds and clouds and dreams that shun the sun,\nWith ba
re boughs rattling in the lonesome winds,\nAnd the dark woodlands brooding over all,\nNot even lig
htened by the rare dim sun\nWhich made squat shadows out of men; they called it\nCimmeria, land of
Darkness and deep Night.\n\n\nIt was so long ago and far away\nI have forgot the very name men cal
led me.\nThe axe and flint-tipped spear are like a dream,\nAnd hunts and wars are shadows. I recal
l\nOnly the stillness of that sombre land;\nThe clouds that piled forever on the hills,\nThe dimne
ss of the everlasting woods.\nCimmeria, land of Darkness and the Night.\n\n\nOh, soul of mine, bor
n out of shadowed hills,\nTo clouds and winds and ghosts that shun the sun,\nHow many deaths shall
serve to break at last\nThis heritage which wraps me in the grey\nApparel of ghosts? I search my h
eart and find\nCimmeria, land of Darkness and the Night."
>>> 
```

STEP 4

If you enter poem.read() a second time you will notice that the text has been removed from the file. You will need to enter: `poem=open("/home/pi/Documents/Poem.txt")` again to recreate the file. This time, however, enter:

```
print(poem.read())
```

This time, the /n entries are removed in favour of new lines and readable text.



```
>>> poem.read()
''
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> print(poem.read())
I remember
The dark woods, masking slopes of sombre hills:
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.


Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.


It was a gloomy land that seemed to hold
All winds and clouds and dreams that shun the sun,
With bare boughs rattling in the lonesome winds,
And the dark woodlands brooding over all,
Not even lightened by the rare dim sun
Which made squat shadows out of men; they called it
Cimmeria, land of Darkness and deep Night.
```

STEP 5

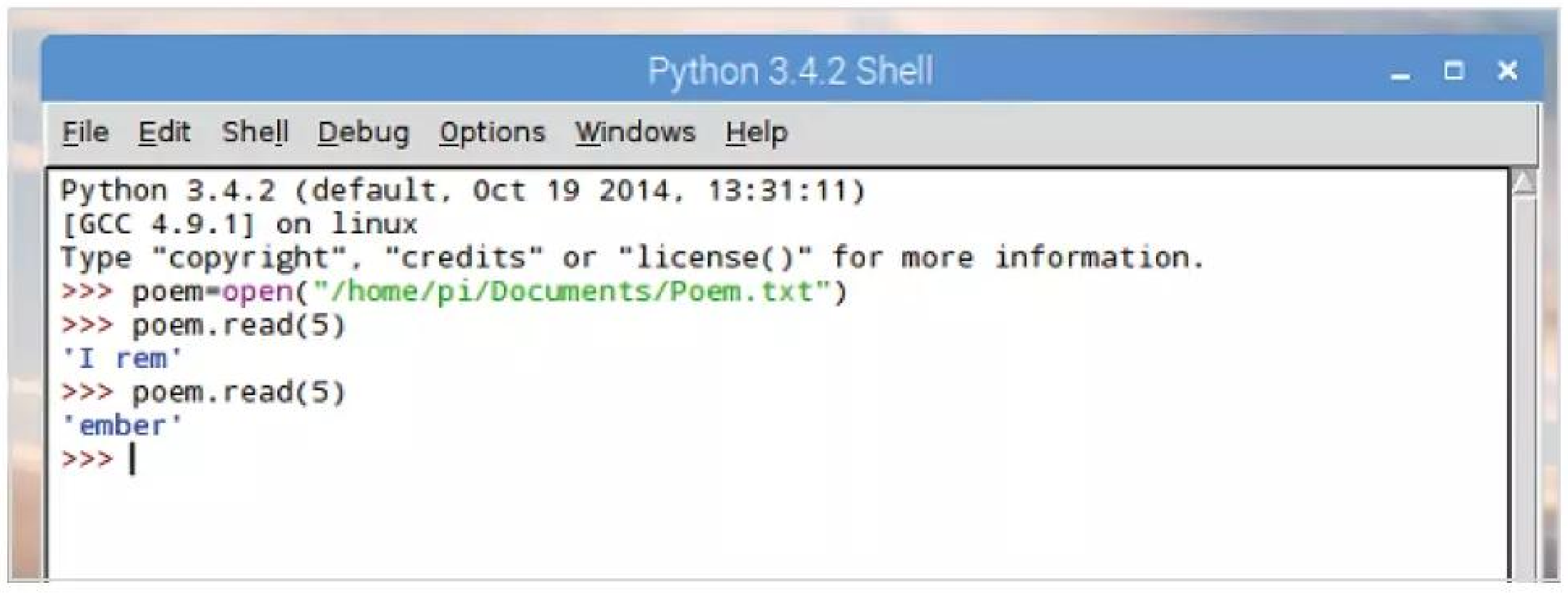
Just as with lists, tuples, dictionaries and so on, you're able to index individual characters of the text. For example:

```
poem.read(5)
```

Displays the first five characters, whilst again entering:

```
poem.read(5)
```

Will display the next five. Entering (1) will display one character at a time.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem.read(5)
'I rem'
>>> poem.read(5)
'ember'
>>> |
```

STEP 6

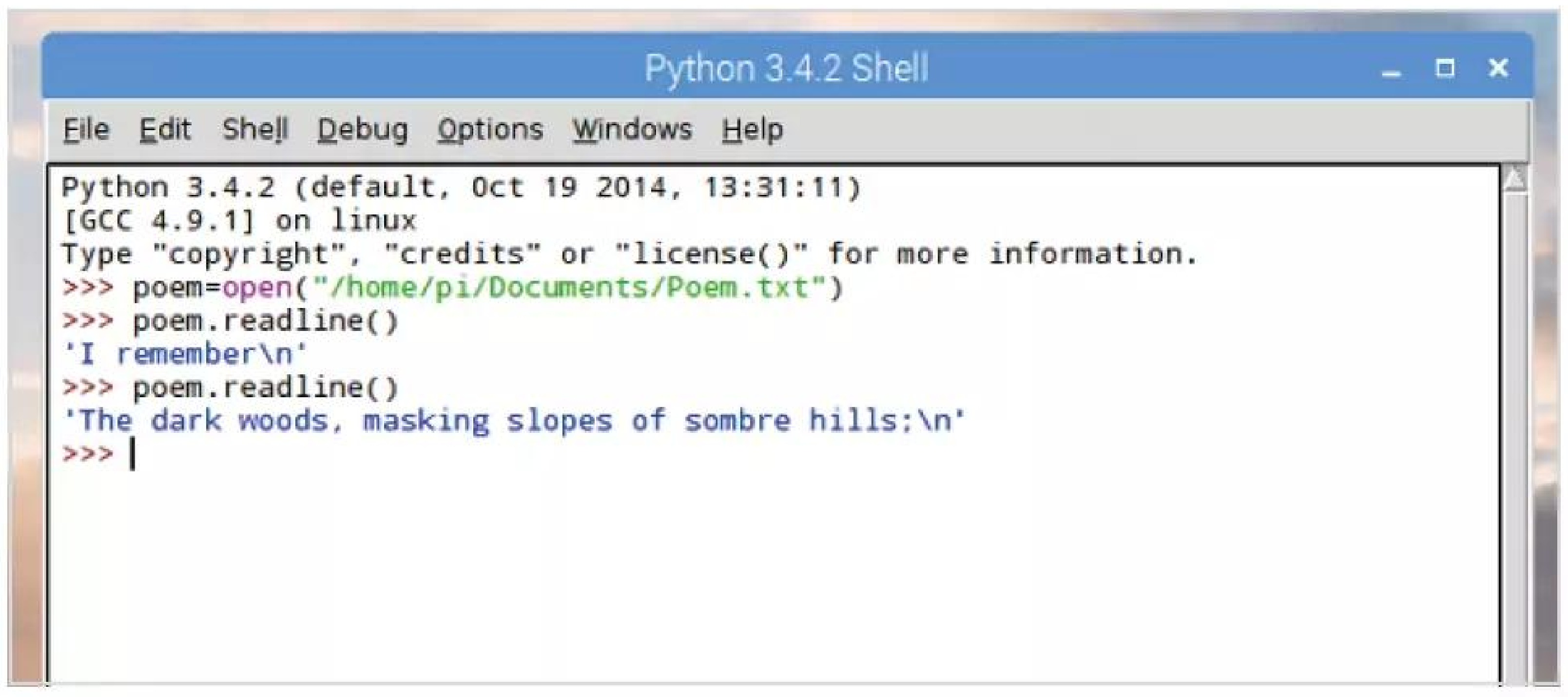
Similarly, you can display one line of text at a time by using the readline() function. For example:

```
poem=open(“/home/pi/Documents/Poem.txt”)
poem.readline()
```

Will display the first line of the text with:

```
poem.readline()
```

Displaying the next line of text once more.

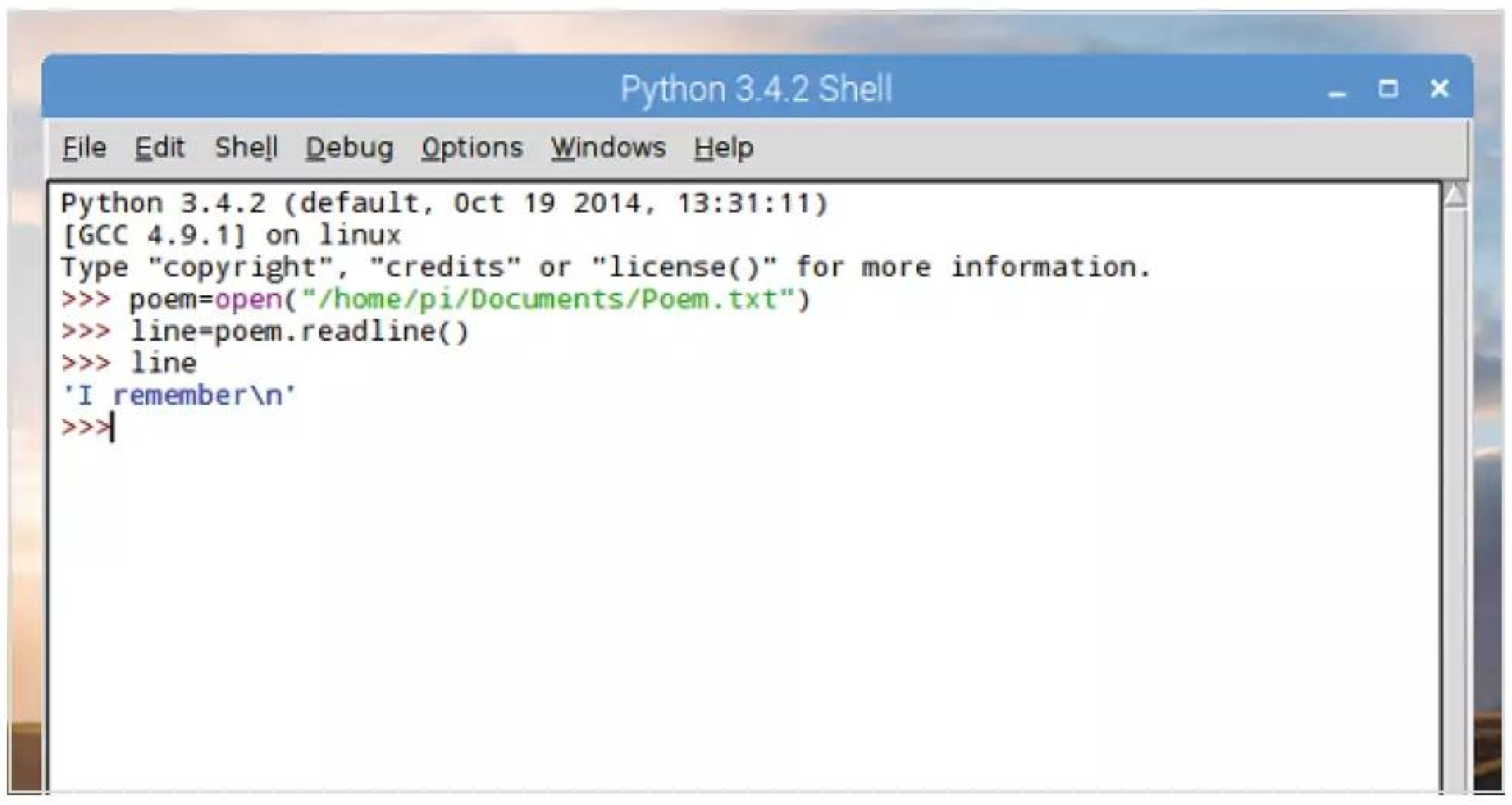

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> poem.readline()
'I remember\n'
>>> poem.readline()
'The dark woods, masking slopes of sombre hills;\n'
>>> |
```

STEP 7

You may have guessed that you can pass the readline() function into a variable, thus allowing you to call it again when needed:

```
poem=open(“/home/pi/Documents/Poem.txt”)
line=poem.readline()
line
```

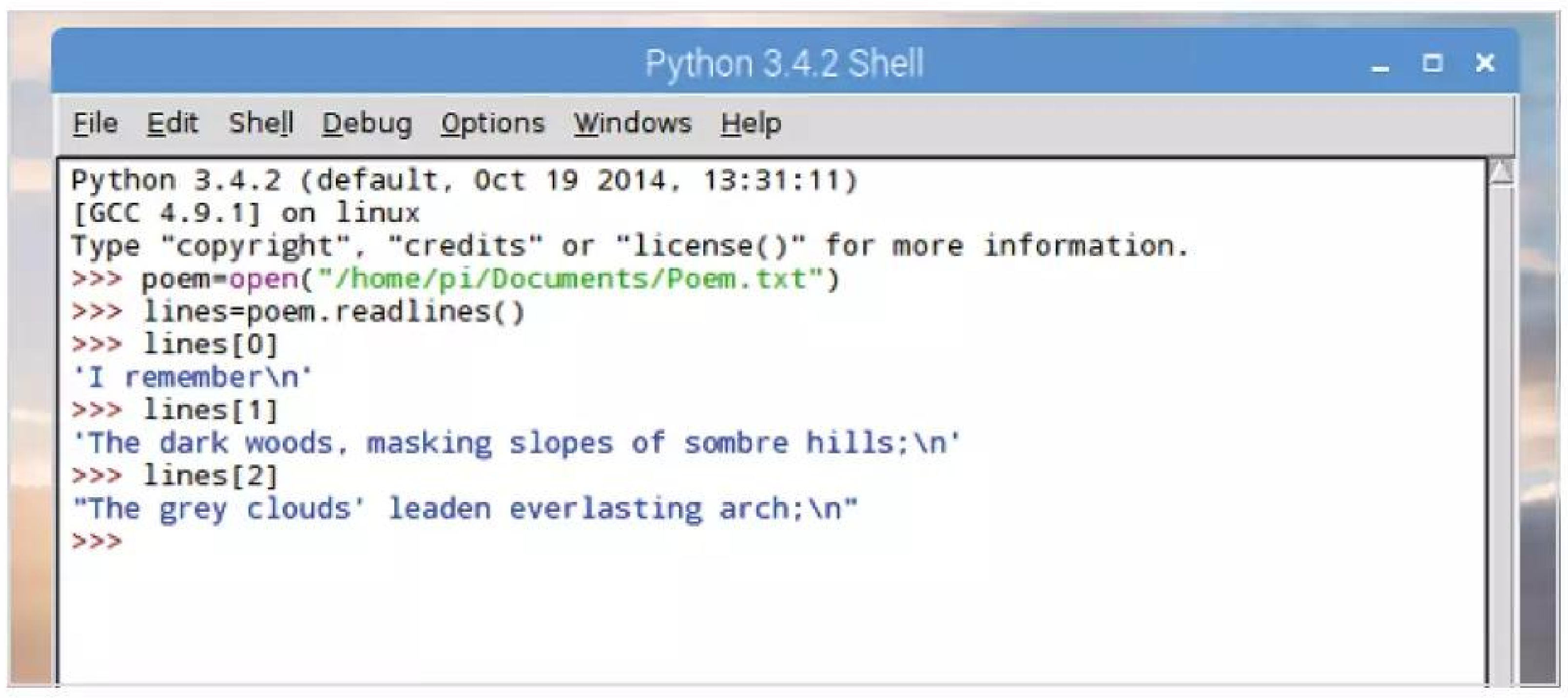

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> line=poem.readline()
>>> line
'I remember\n'
>>>|
```

STEP 8

Extending this further, you can use readlines() to grab all the lines of the text and store them as multiple lists. These can then be stored as a variable:

```
poem=open(“/home/pi/Documents/Poem.txt”)
lines=poem.readlines()
lines[0]
lines[1]
lines[2]
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> lines=poem.readlines()
>>> lines[0]
'I remember\n'
>>> lines[1]
'The dark woods, masking slopes of sombre hills;\n'
>>> lines[2]
"The grey clouds' leaden everlasting arch;\n"
>>>
```

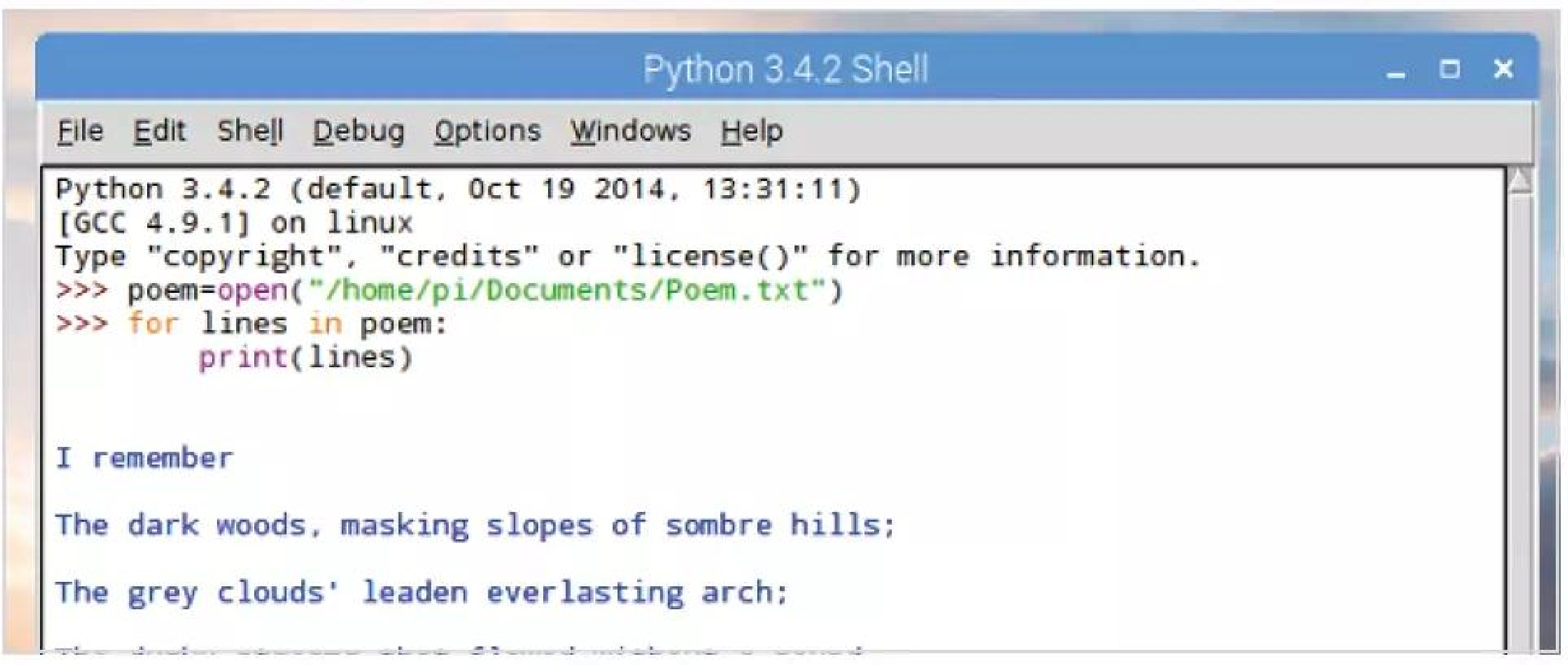
STEP 9

You can also use the for statement to read the lines of text back to us:

```
for lines in lines:
   print(lines)
```

Since this is Python, there are other ways to produce the same output:

```
poem=open(“/home/pi/Documents/Poem.txt”)
for lines in poem:
   print(lines)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

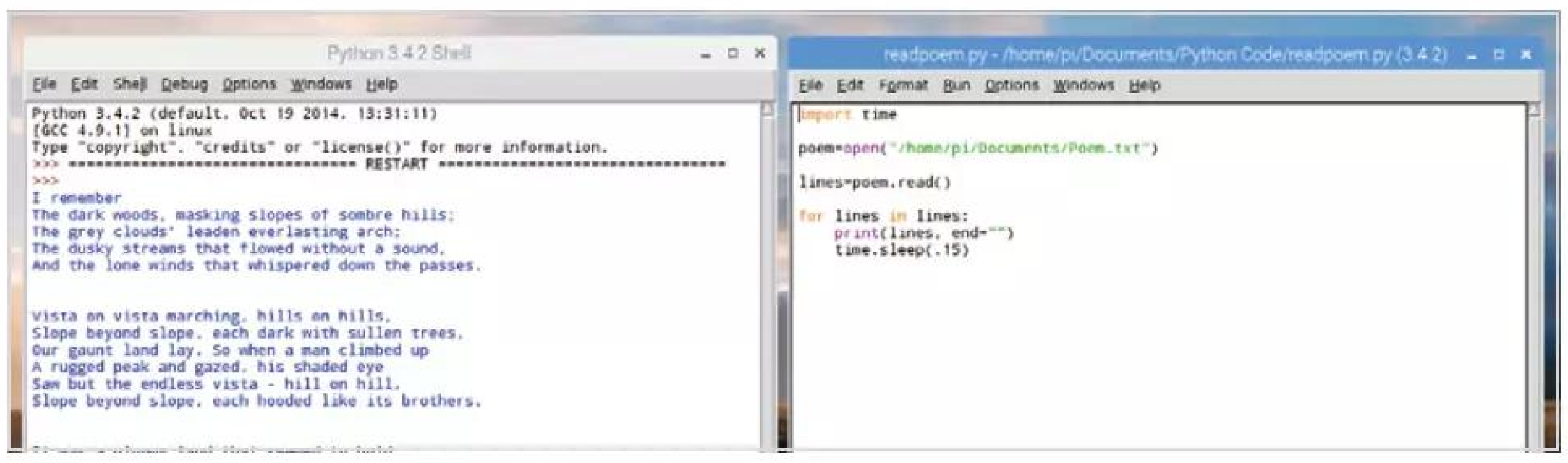
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> poem=open("/home/pi/Documents/Poem.txt")
>>> for lines in poem:
        print(lines)


I remember

The dark woods, masking slopes of sombre hills;

The grey clouds' leaden everlasting arch;
```

STEP 10

Let's imagine that you want to print the text one character at a time, like an old dot matrix printer would. You can use the time module mixed with what you've looked at here. Try this:

```
import time
poem=open(“/home/pi/Documents/Poem.txt”)
lines=poem.read()
for lines in lines:
  print(lines, end=””)
  time.sleep(.15)
```

The output is fun to view, and easily incorporated into your own code.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
readpoem.py - /home/pi/Documents/Python Code/readpoem.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
I remember
The dark woods, masking slopes of sombre hills;
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.


Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.
```

```
import time

poem=open("/home/pi/Documents/Poem.txt")

lines=poem.read()

for lines in lines:
    print(lines, end="")
    time.sleep(.15)
```

Writing to Files

The ability to read external files within Python is certainly handy but writing to a file is better still. Using the write() function, you're able to output the results of a program to a file, that you can then read() back into Python.

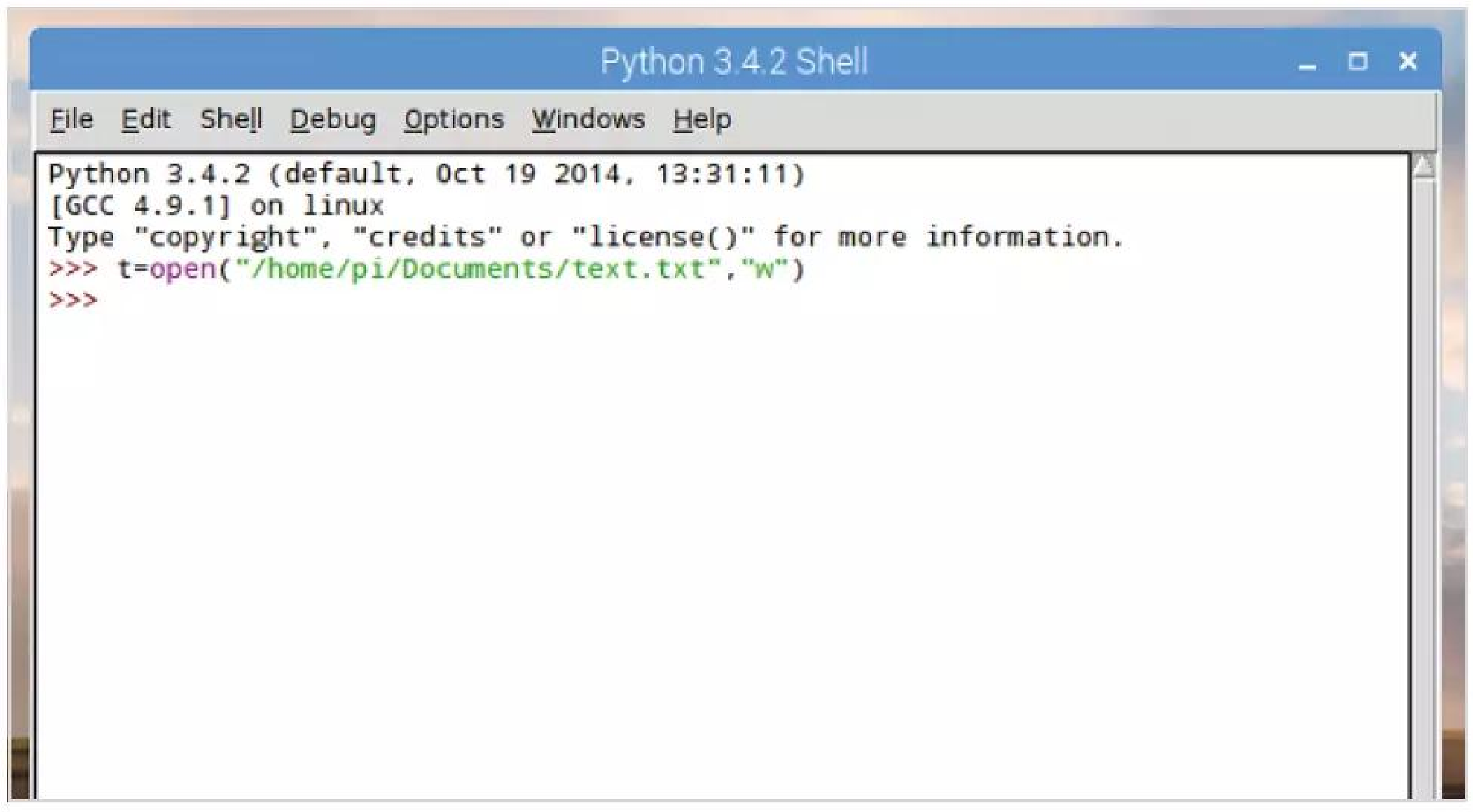
WRITE AND CLOSE

The write() function is slightly more complex than read(). Along with the filename you must also include an access mode which determines whether the file in question is in read or write mode.

STEP 1 Start by opening IDLE and enter the following:

```
t=open(“/home/pi/Documents/text.txt”,”w”)
```

Change the destination from /home/pi/Documents to your own system. This code will create a text file called text.txt in write mode using the variable 't'. If there's no file of that name in the location, it will create one. If one already exits, it will overwrite it, so be careful.

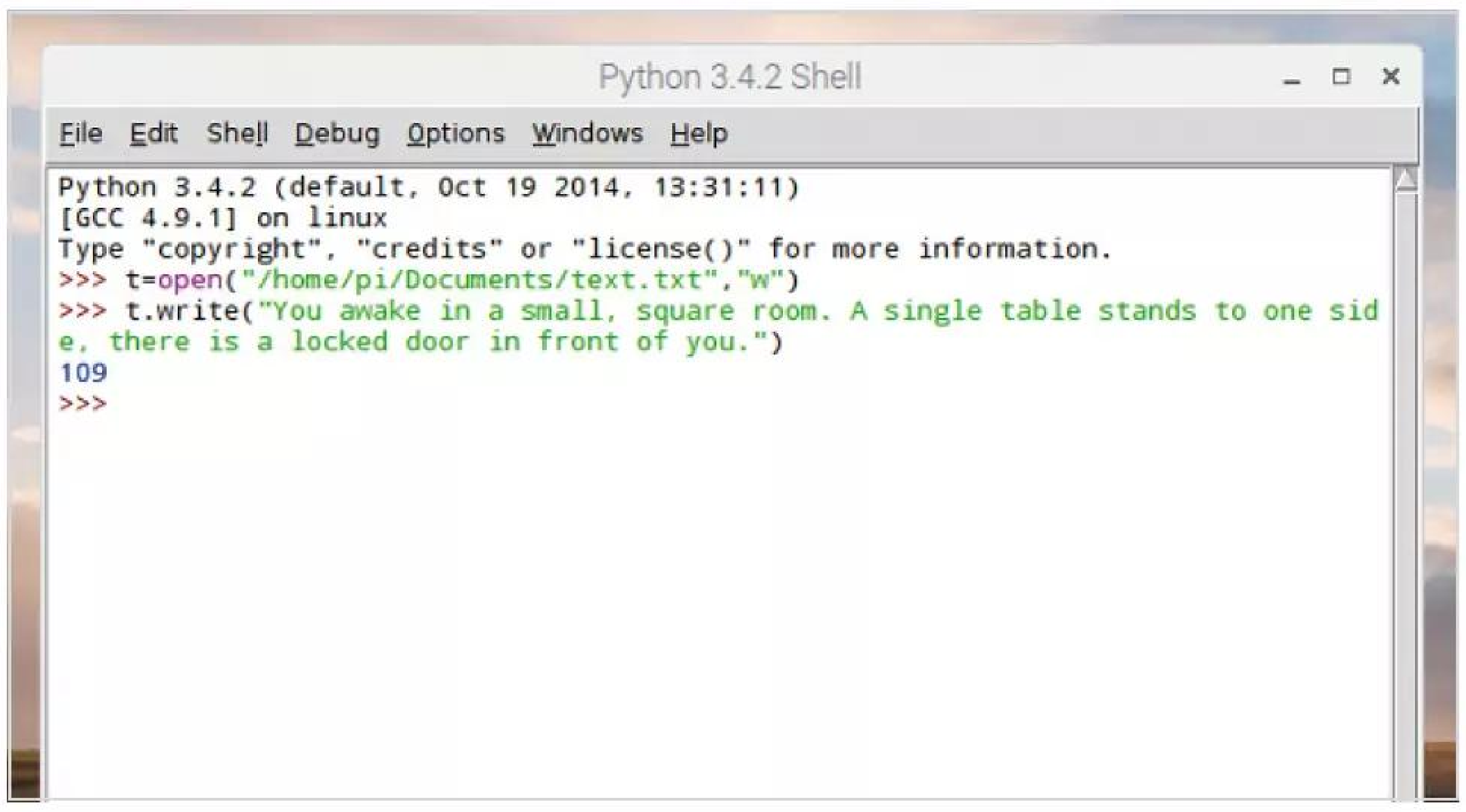


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>>
```

STEP 2 You can now write to the text file using the write() function. This works opposite to read(), writing lines instead of reading them. Try this:

```
t.write(“You awake in a small, square room. A single table stands to one side, there is a locked door in front of you.”)
```

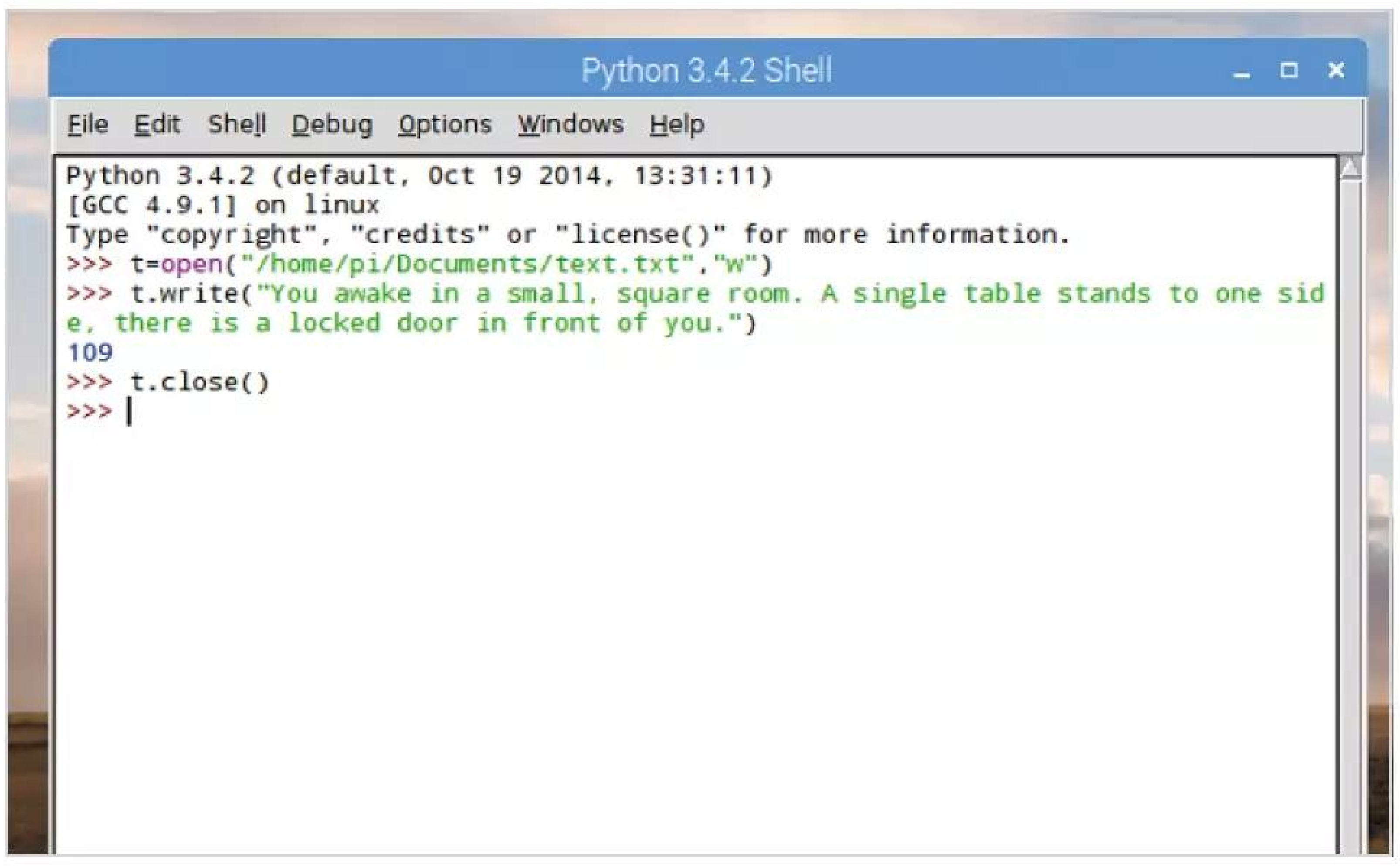
Note the 109. It's the number of characters you've entered.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>> t.write("You awake in a small, square room. A single table stands to one sid
e, there is a locked door in front of you.")
109
>>>
```

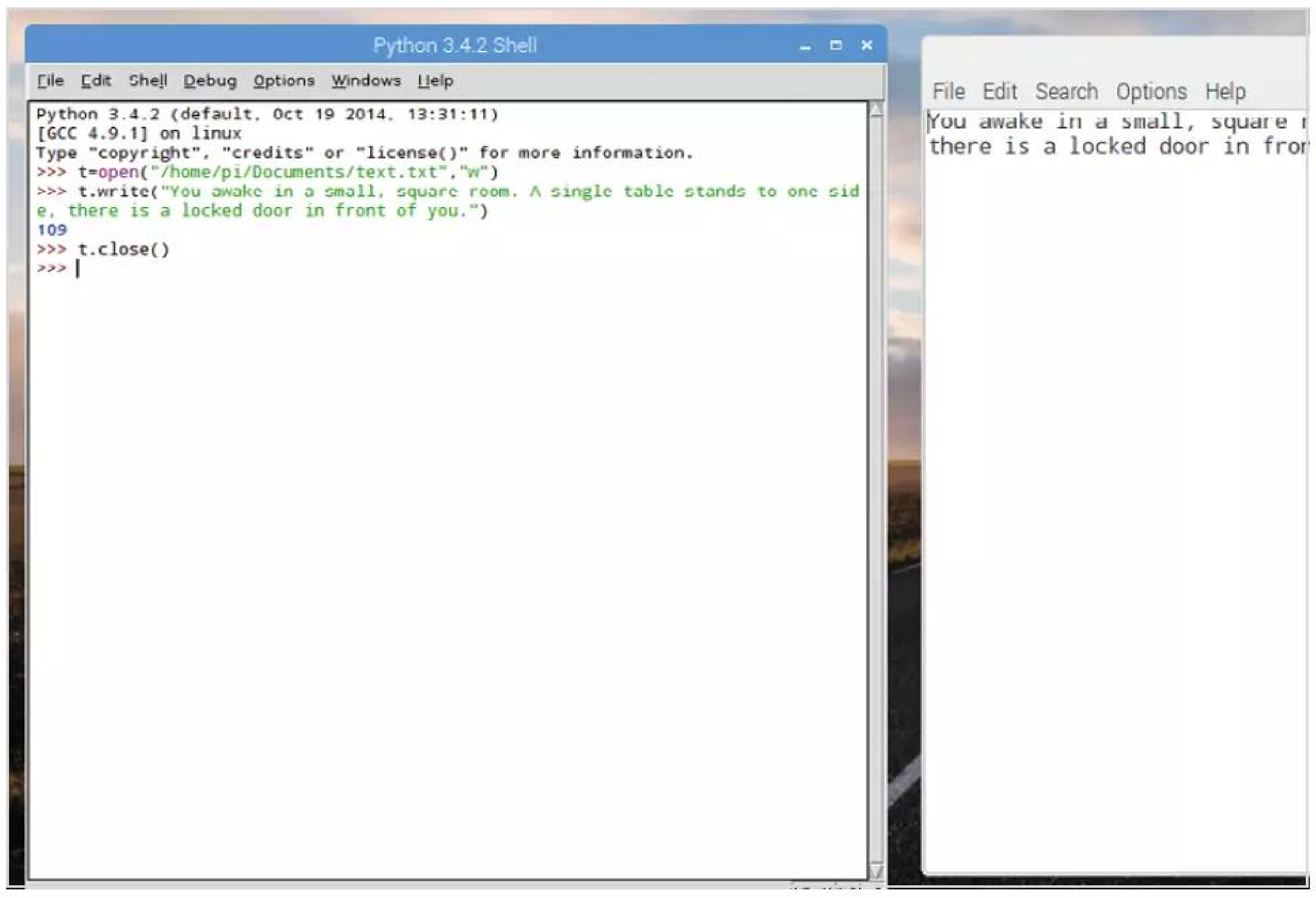
STEP 3 However, the actual text file is still blank (you can check by opening it up). This is because you've written the line of text to the file object but not committed it to the file itself. Part of the write() function is that you need to commit the changes to the file; you can do this by entering:

```
t.close()
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>> t.write("You awake in a small, square room. A single table stands to one sid
e, there is a locked door in front of you.")
109
>>> t.close()
>>>
```

STEP 4 If you now open the text file with a text editor, you can see that the line you created has been written to the file. This gives us the foundation for some interesting possibilities: perhaps the creation of your own log file or even the beginning of an adventure game.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
File Edit Search Options Help

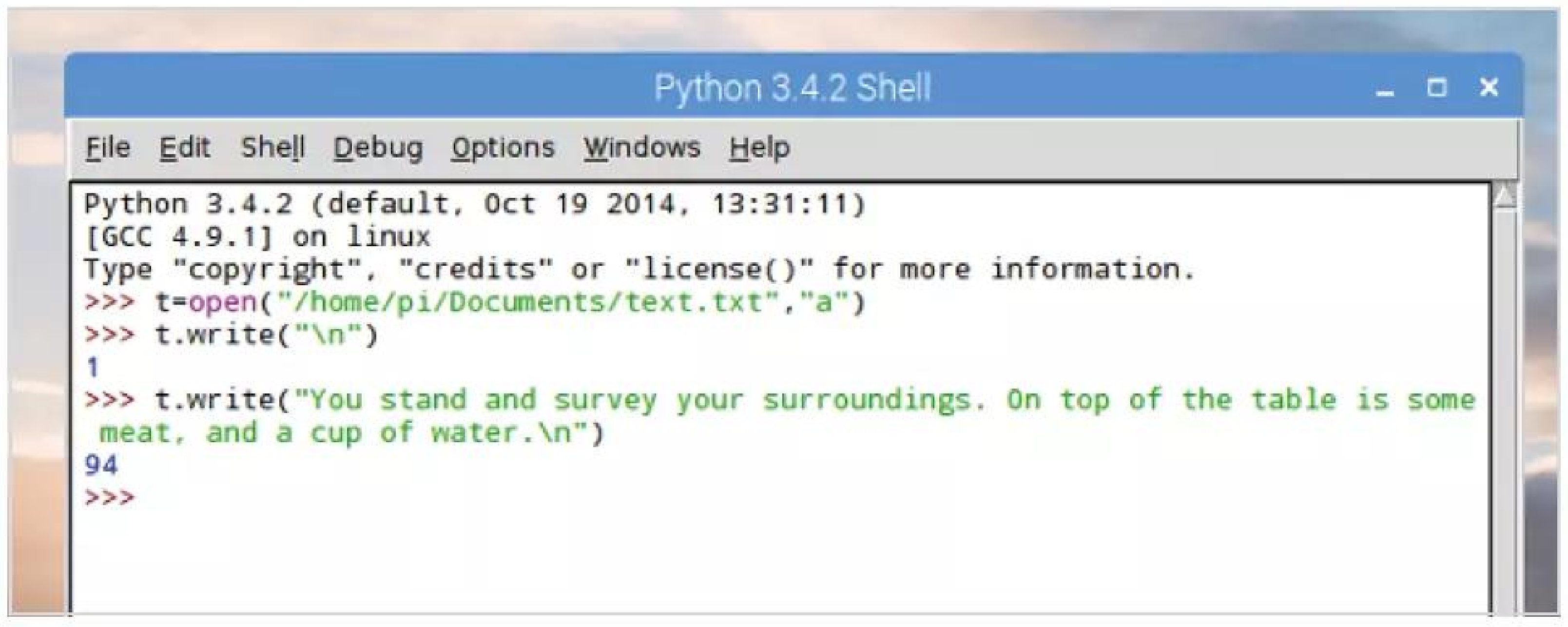
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","w")
>>> t.write("You awake in a small, square room. A single table stands to one sid
e, there is a locked door in front of you.")
109
>>> t.close()
>>>
```

```
You awake in a small, square r
there is a locked door in fror
```

STEP 5

To expand this code, you can reopen the file using 'a', for access or append mode. This will add any text at the end of the original line instead of wiping the file and creating a new one. For example:

```
t=open("/home/pi/Documents/text.txt","a")
t.write("\n")
t.write(" You stand and survey your surroundings. On top of the table is some meat, and a cup of water.\n")
```

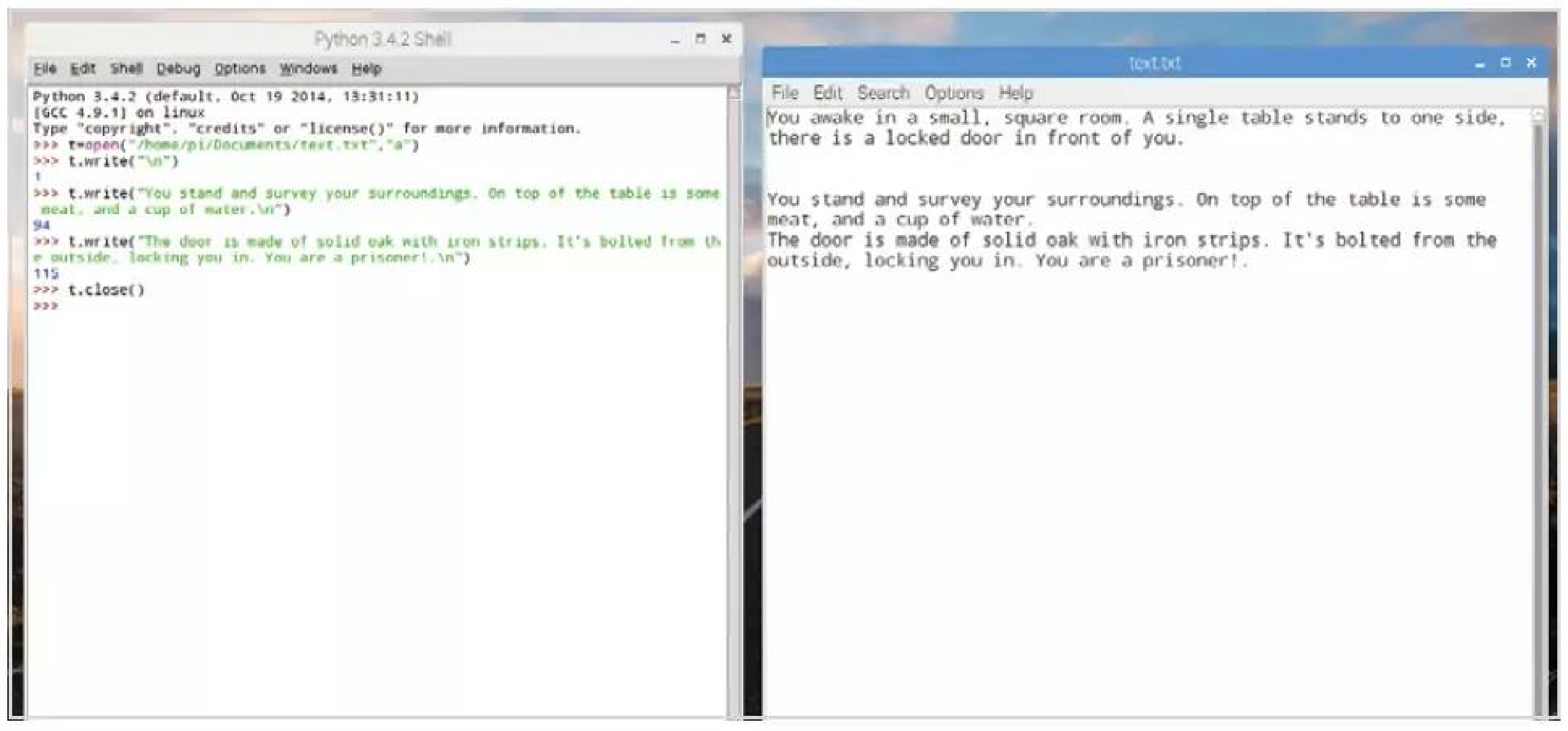

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","a")
>>> t.write("\n")
1
>>> t.write("You stand and survey your surroundings. On top of the table is some
 meat, and a cup of water.\n")
94
>>>
```

STEP 6

You can keep extending the text line by line, ending each with a new line (\n). When you're done, finish the code with t.close() and open the file in a text editor to see the results:

```
t.write("The door is made of solid oak with iron strips. It's bolted from the outside, locking you in. You are a prisoner!.\n")
t.close()
```

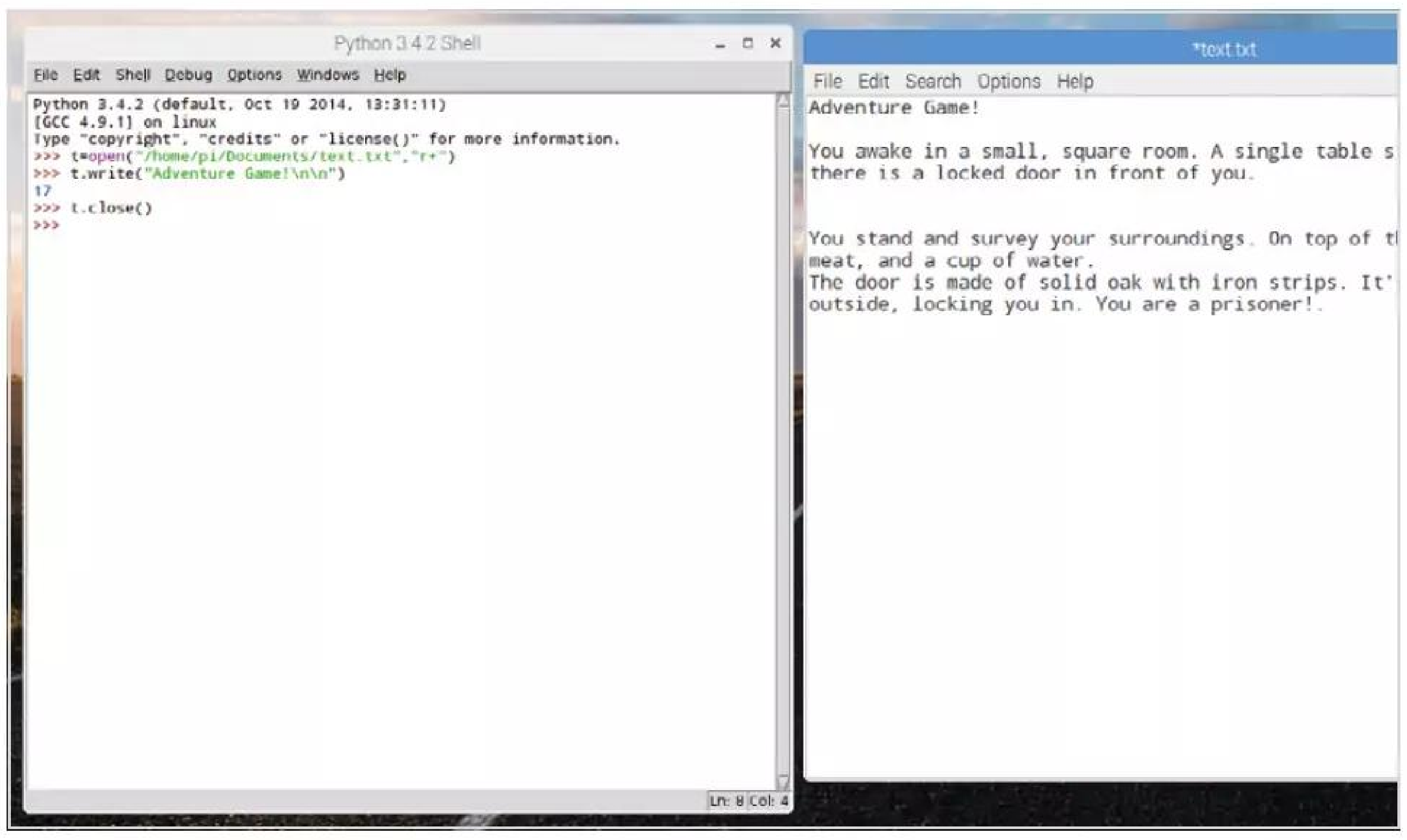

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
text.txt
File Edit Search Options Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","a")
>>> t.write("\n")
1
>>> t.write("You stand and survey your surroundings. On top of the table is some
 meat, and a cup of water.\n")
94
>>> t.write("The door is made of solid oak with iron strips. It's bolted from th
e outside, locking you in. You are a prisoner!.\n")
115
>>> t.close()
>>>
```

```
You awake in a small, square room. A single table stands to one side,
there is a locked door in front of you.


You stand and survey your surroundings. On top of the table is some
meat, and a cup of water.
The door is made of solid oak with iron strips. It's bolted from the
outside, locking you in. You are a prisoner!.
```

STEP 7

There are various types of file access to consider using the open() function. Each depends on how the file is accessed and even the position of the cursor. For example, r+ opens a file in read and write and places the cursor at the start of the file.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
*text.txt
File Edit Search Options Help
Ln: 8 Col: 4

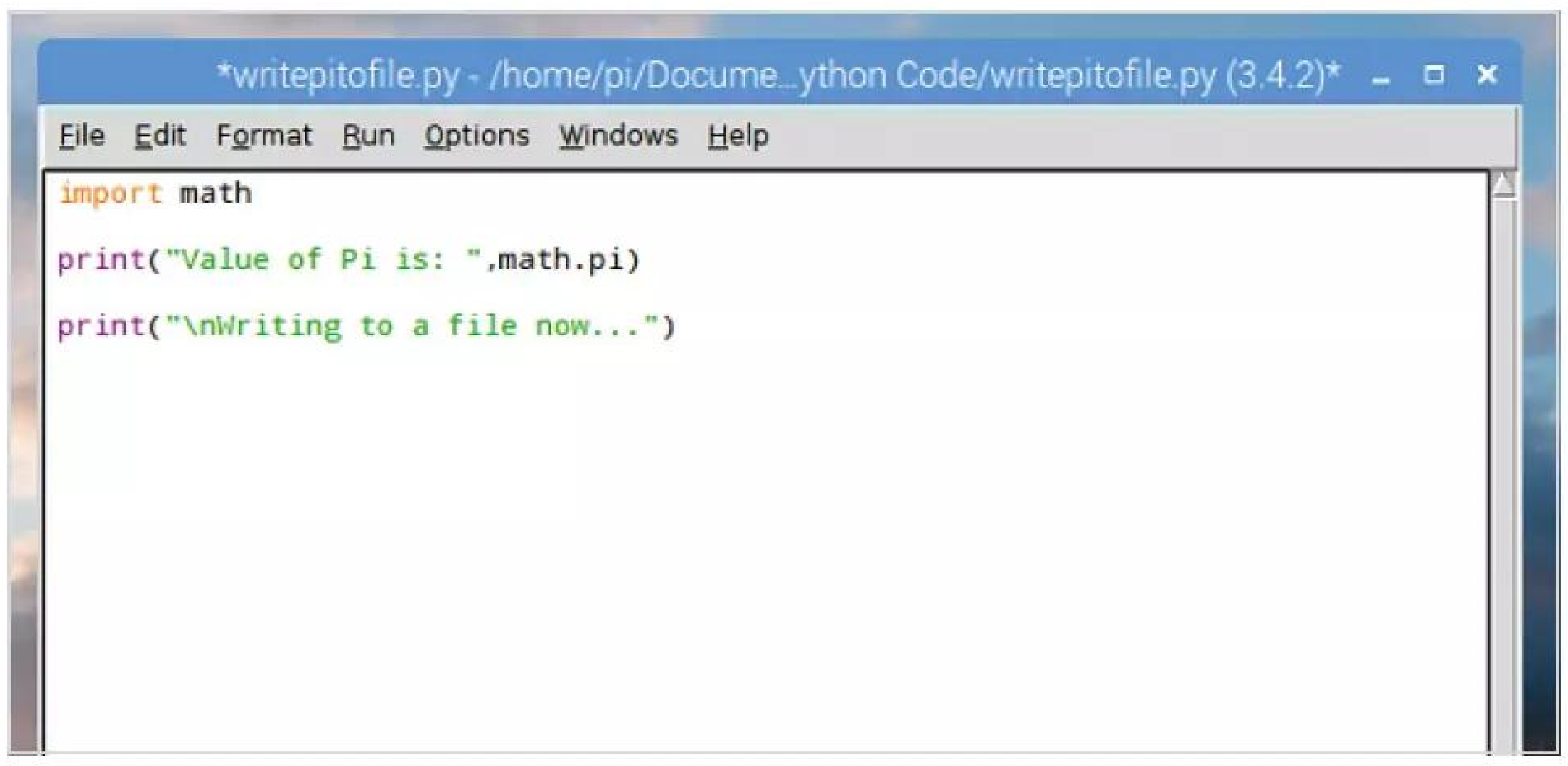
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> t=open("/home/pi/Documents/text.txt","r+")
>>> t.write("Adventure Game!\n\n")
17
>>> t.close()
>>>
```

```
Adventure Game!

You awake in a small, square room. A single table s
there is a locked door in front of you.


You stand and survey your surroundings. On top of t
meat, and a cup of water.
The door is made of solid oak with iron strips. It'
outside, locking you in. You are a prisoner!.
```

STEP 8

You can pass variables to a file that you've created in Python. Perhaps you want the value of Pi to be written to a file. You can call Pi from the math module, create a new file and pass the output of Pi into the new file:

```
import math
print("Value of Pi is: ",math.pi)
print("\nWriting to a file now…")
```


writepitofile.py - /home/pi/Docume...ython Code/writepitofile.py (3.4.2)
File Edit Format Run Options Windows Help

```
import math

print("Value of Pi is: ",math.pi)

print("\nWriting to a file now...")
```

STEP 9

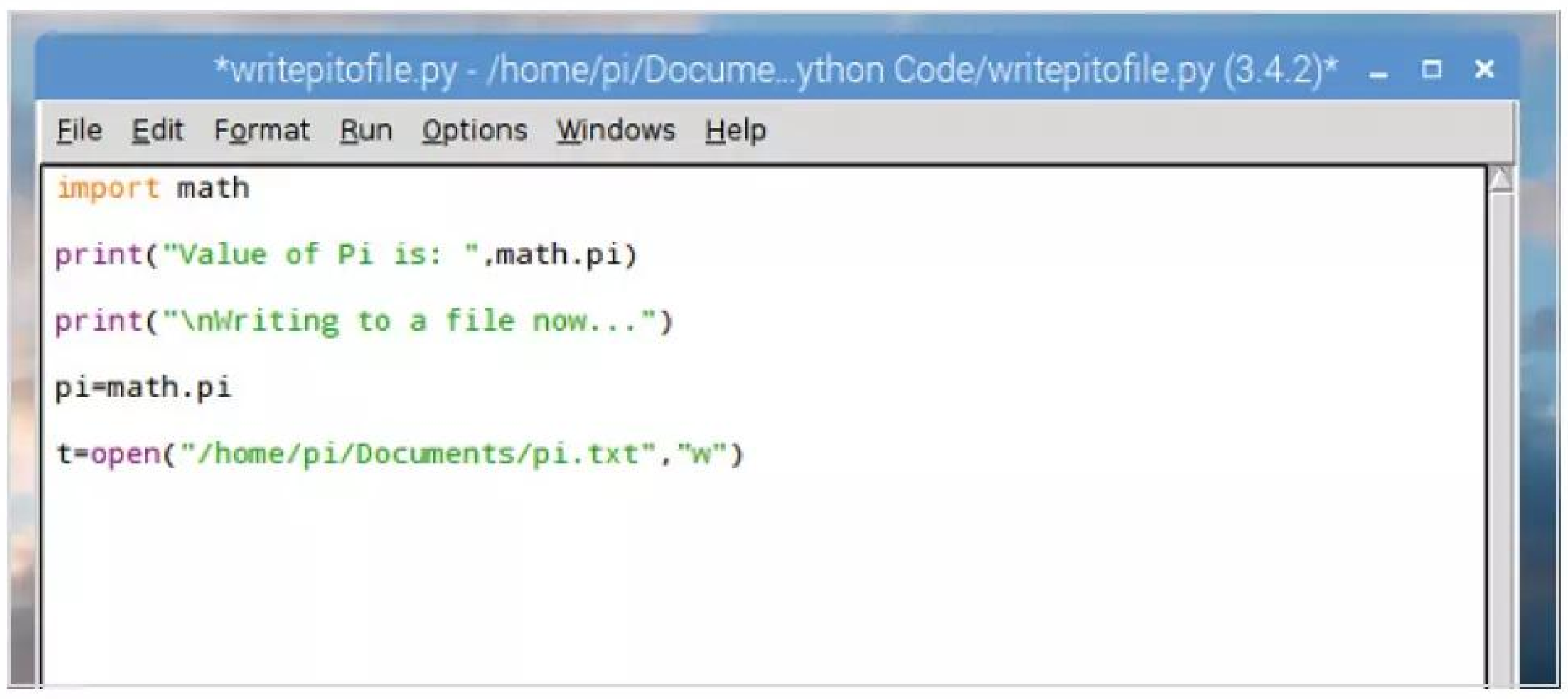
Now let's create a variable called pi and assign it the value of Pi:

```
pi=math.pi
```

You also need to create a new file in which to write Pi to:

```
t=open("/home/pi/Documents/pi.txt","w")
```

Remember to change your file location to your own particular system setup.


writepitofile.py - /home/pi/Docume...ython Code/writepitofile.py (3.4.2)
File Edit Format Run Options Windows Help

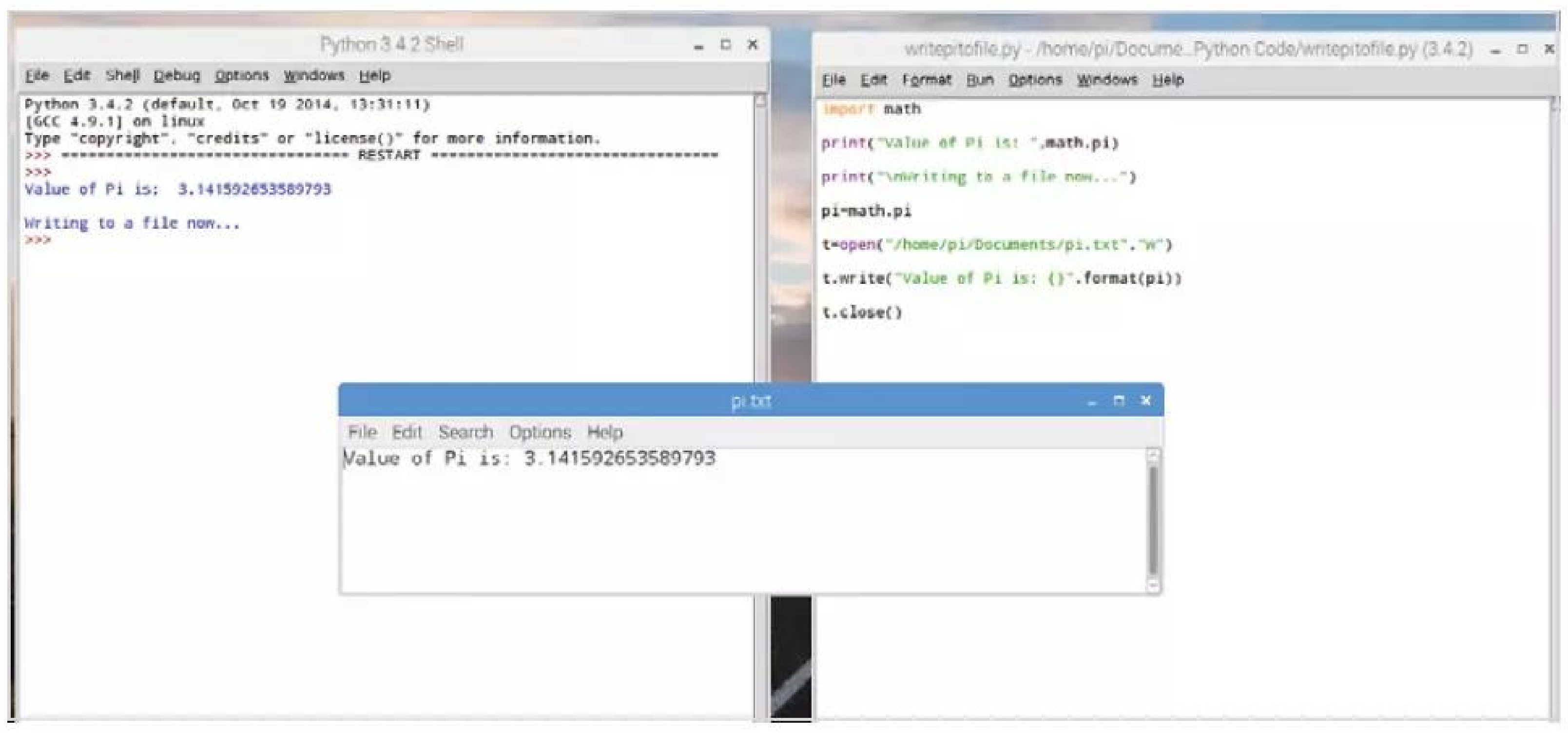
```
import math

print("Value of Pi is: ",math.pi)

print("\nWriting to a file now...")

pi=math.pi

t=open("/home/pi/Documents/pi.txt","w")
```

STEP 10

To finish, you can use string formatting to call the variable and write it to the file, then commit the changes and close the file:

```
t.write("Value of Pi is: {}".format(pi))
t.close()
```

You can see from the results that you're able to pass any variable to a file.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
writepitofile.py - /home/pi/Docume...Python Code/writepitofile.py (3.4.2)
File Edit Format Run Options Windows Help
pi.txt
File Edit Search Options Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Value of Pi is:  3.141592653589793

Writing to a file now...
>>>
```

```
import math

print("Value of Pi is: ",math.pi)

print("\nWriting to a file now...")

pi=math.pi

t=open("/home/pi/Documents/pi.txt","w")

t.write("Value of Pi is: {}".format(pi))

t.close()
```

```
Value of Pi is: 3.141592653589793
```

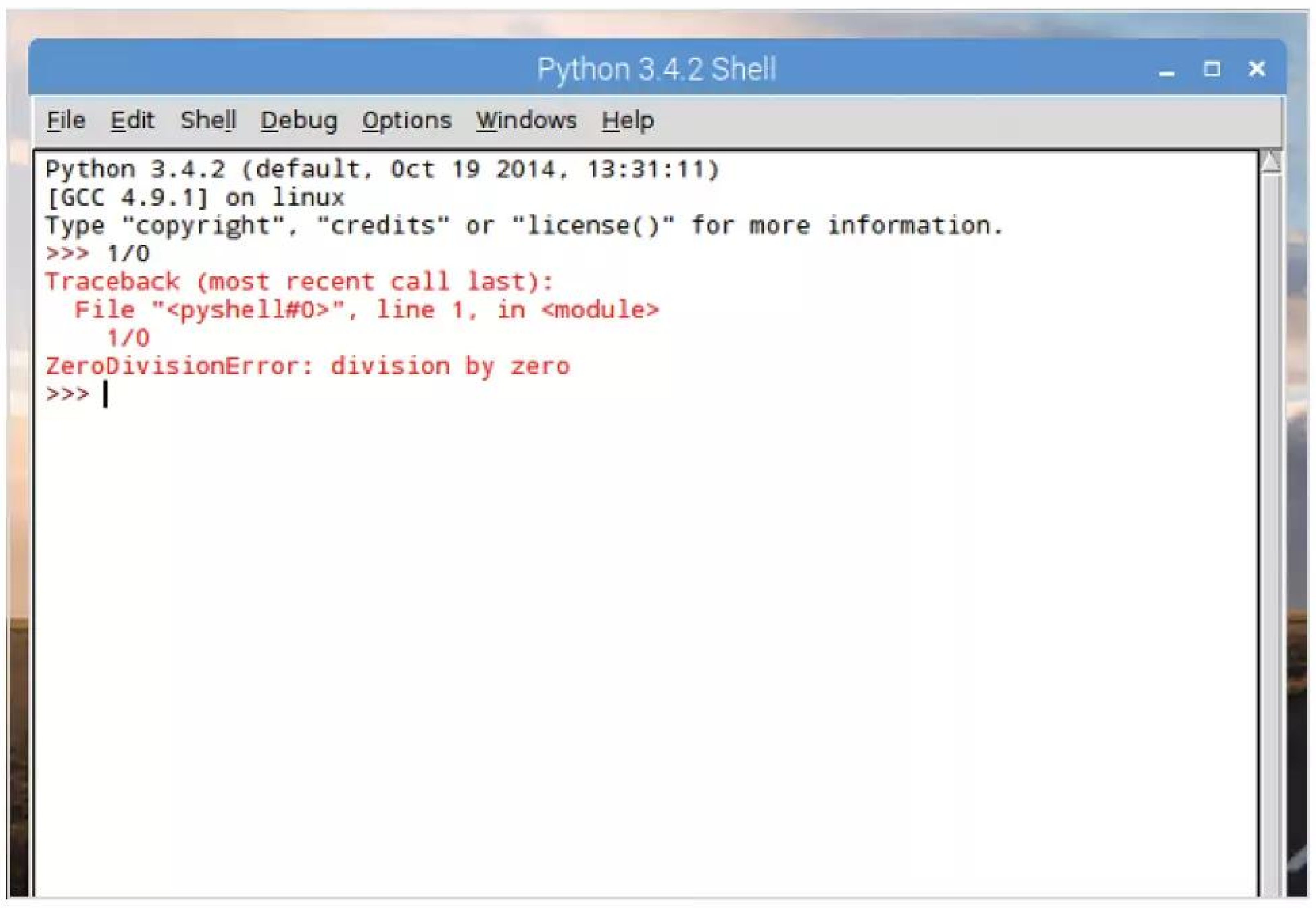
Exceptions

When coding, you'll naturally come across some issues that are out of your control. Let's assume you ask a user to divide two numbers and they try to divide by zero. This will create an error and break your code.

EXCEPTIONAL OBJECTS

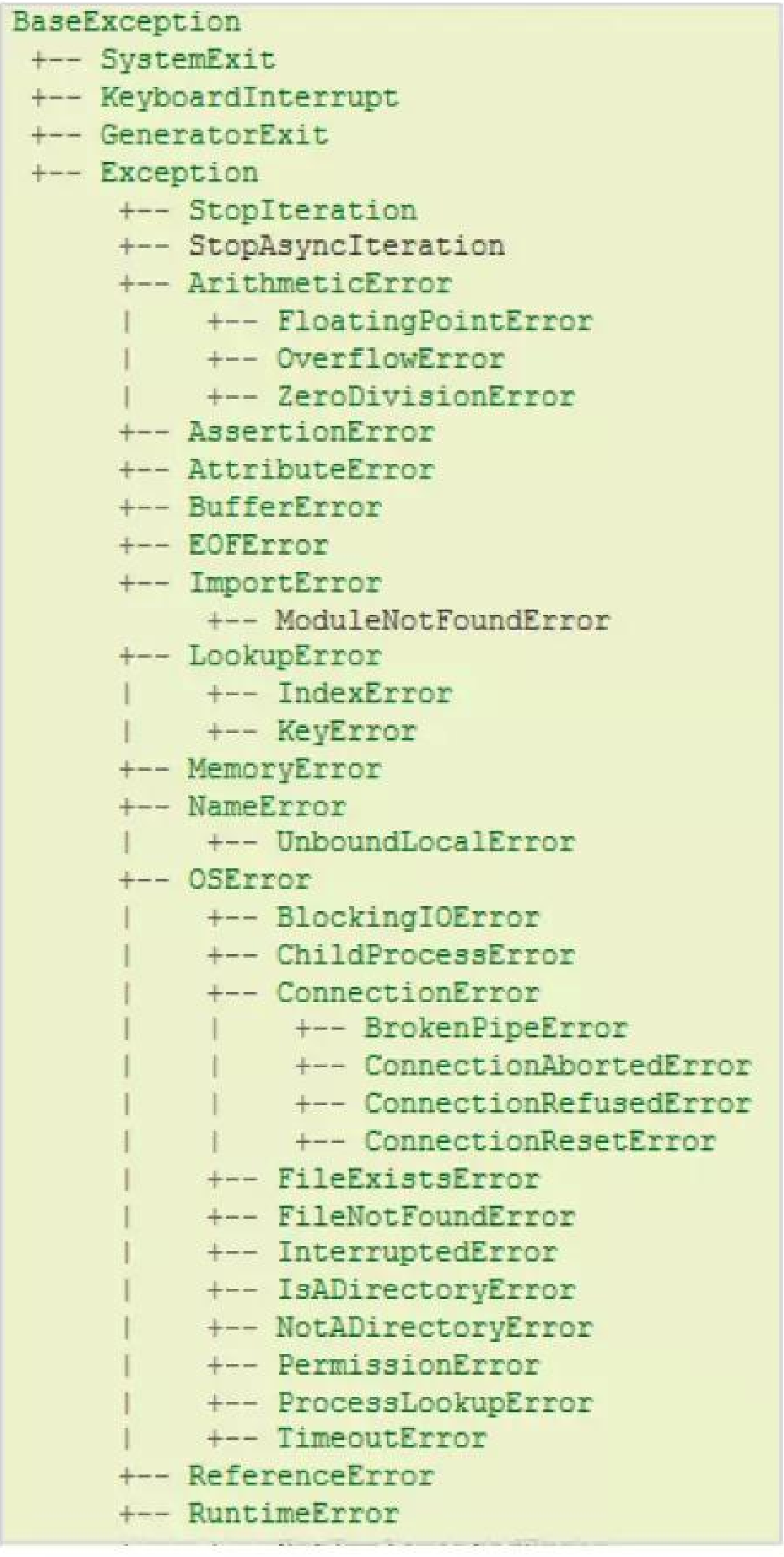
Rather than stop the flow of your code, Python includes exception objects which handle unexpected errors in the code. You can combat errors by creating conditions where exceptions may occur.

STEP 1 You can create an exception error by simply trying to divide a number by zero. This will report back with the ZeroDivisionError: Division by zero message, as seen in the screenshot. The ZeroDivisionError part is the exception class, of which there are many.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 1/0
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    1/0
ZeroDivisionError: division by zero
>>> 
```

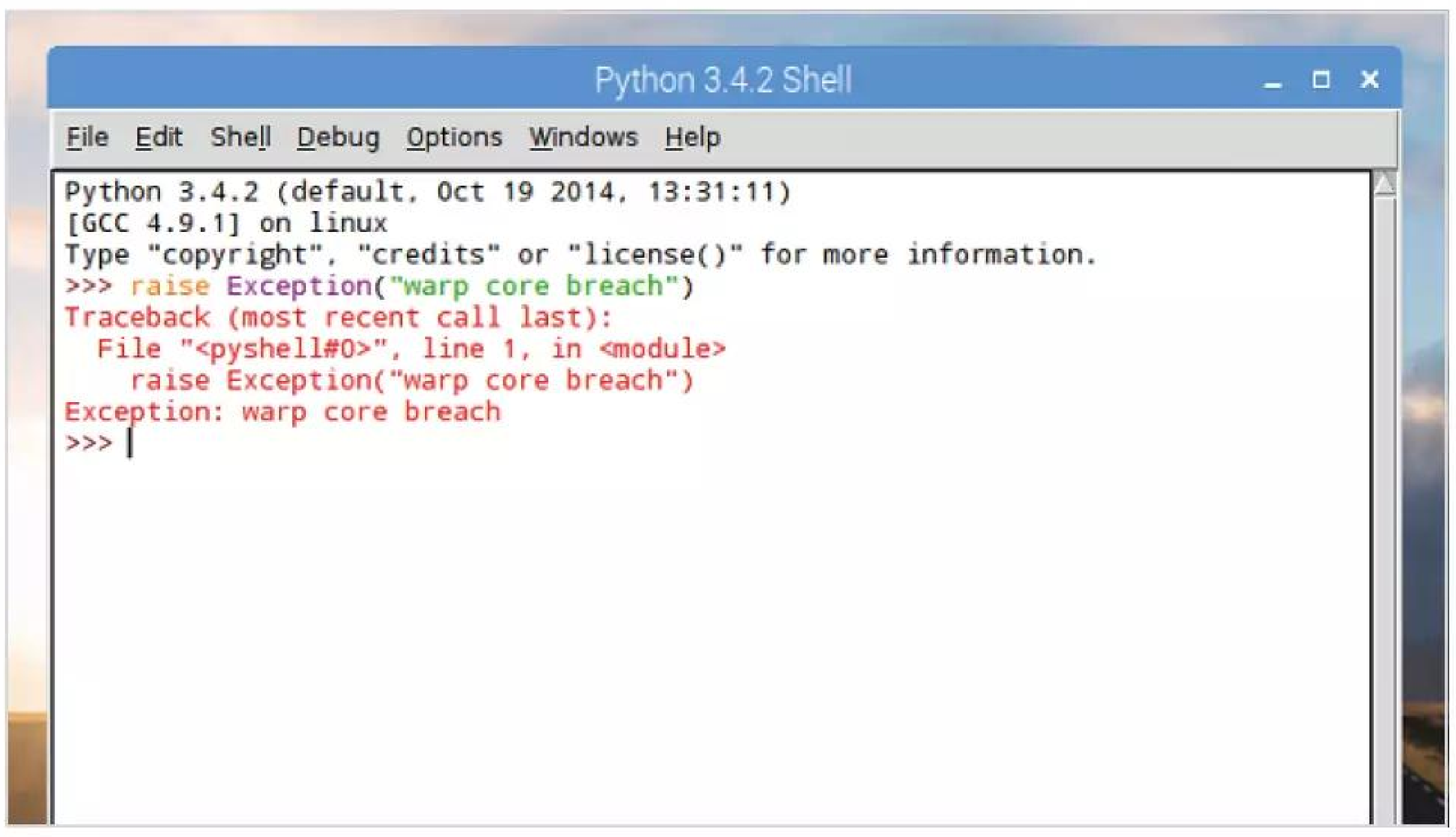
STEP 2 Most exceptions are raised automatically when Python comes across something that's inherently wrong with the code. However, you can create your own exceptions that are designed to contain the potential error and react to it, as opposed to letting the code fail.



```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
           +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
```

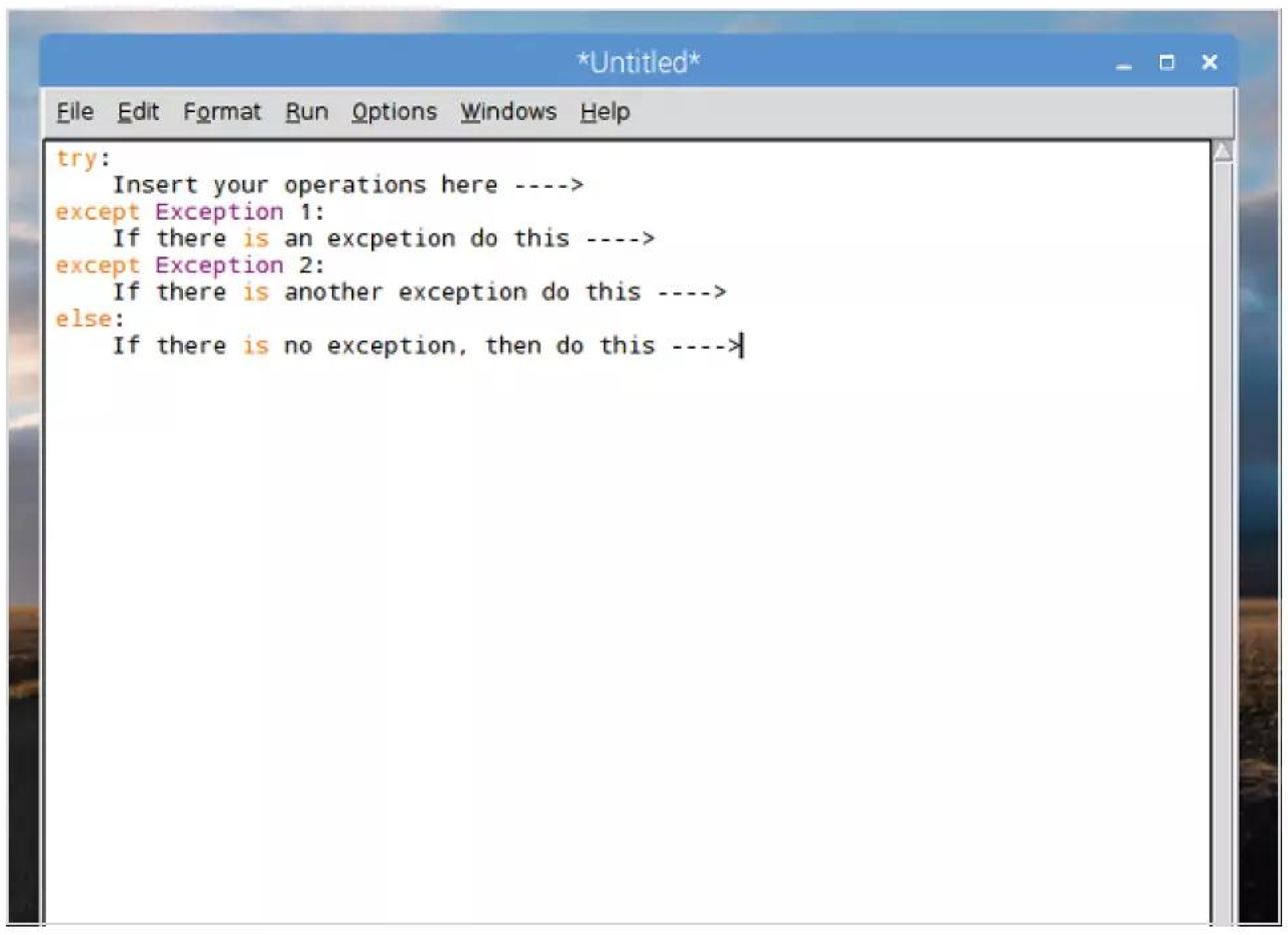
STEP 3 You can use the functions raise exception to create our own error handling code within Python. Let's assume your code has you warping around the cosmos, too much however results in a warp core breach. To stop the game from exiting due to the warp core going supernova, you can create a custom exception:

`raise Exception(“warp core breach”)`



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> raise Exception("warp core breach")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    raise Exception("warp core breach")
Exception: warp core breach
>>> 
```

STEP 4 To trap any errors in the code you can encase the potential error within a try: block. This block consists of try, except, else, where the code is held within try:, then if there's an exception do something, else do something else.

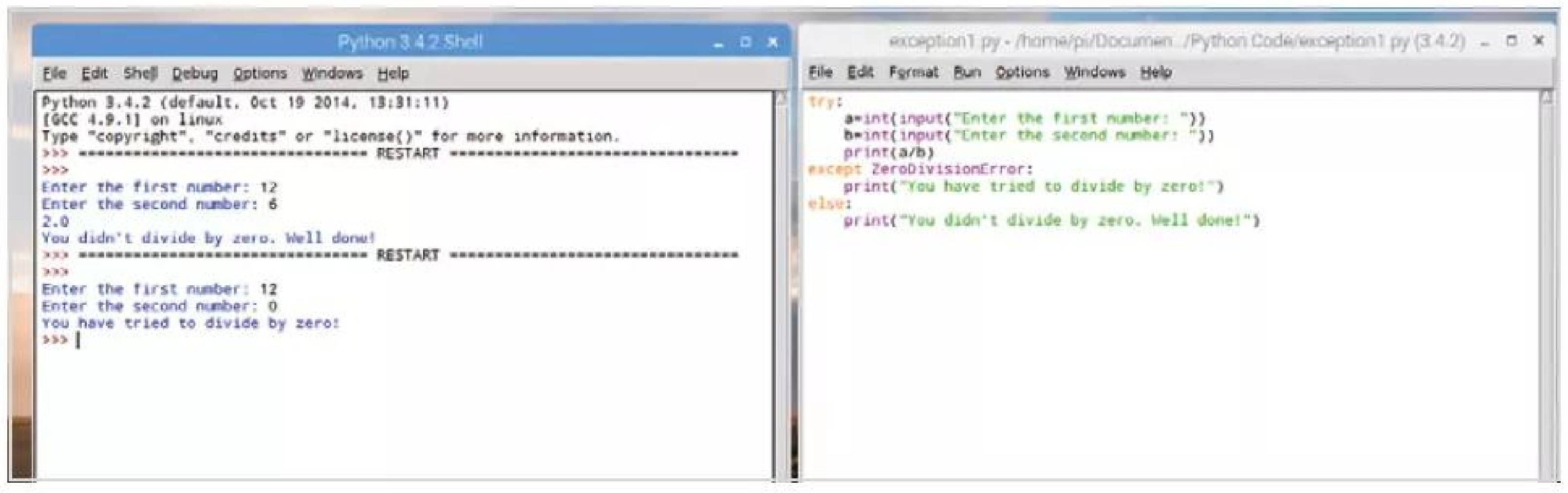


```
try:
    Insert your operations here ---->
except Exception 1:
    If there is an excpetion do this ---->
except Exception 2:
    If there is another exception do this ---->
else:
    If there is no exception, then do this ---->
```

STEP 5

For example, use the divide by zero error. You can create an exception where the code can handle the error without Python quitting due to the problem:

```
try:
  a=int(input("Enter the first number: "))
  b=int(input("Enter the second number: "))
  print(a/b)
except ZeroDivisionError:
  print("You have tried to divide by zero!")
else:
  print("You didn't divide by zero. Well done!")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
exception1.py - /home/pi/Documen.../Python Code/exception1.py (3.4.2)
File Edit Format Run Options Windows Help

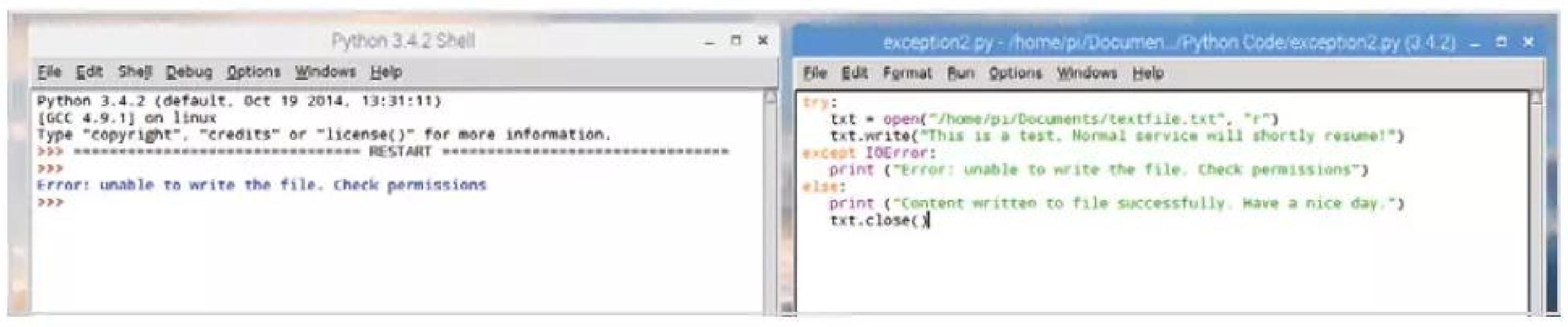
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter the first number: 12
Enter the second number: 6
2.0
You didn't divide by zero. Well done!
>>> ================================ RESTART ================================
>>>
Enter the first number: 12
Enter the second number: 0
You have tried to divide by zero!
>>> |
```

```
try:
    a=int(input("Enter the first number: "))
    b=int(input("Enter the second number: "))
    print(a/b)
except ZeroDivisionError:
    print("You have tried to divide by zero!")
else:
    print("You didn't divide by zero. Well done!")
```

STEP 6

You can use exceptions to handle a variety of useful tasks. Using an example from our previous tutorials, let's assume you want to open a file and write to it:

```
try:
 txt = open("/home/pi/Documents/textfile.txt", "r")
 txt.write("This is a test. Normal service will
shortly resume!")
except IOError:
 print ("Error: unable to write the file. Check
permissions")
else:
 print ("Content written to file successfully. Have
a nice day.")
 txt.close()
```

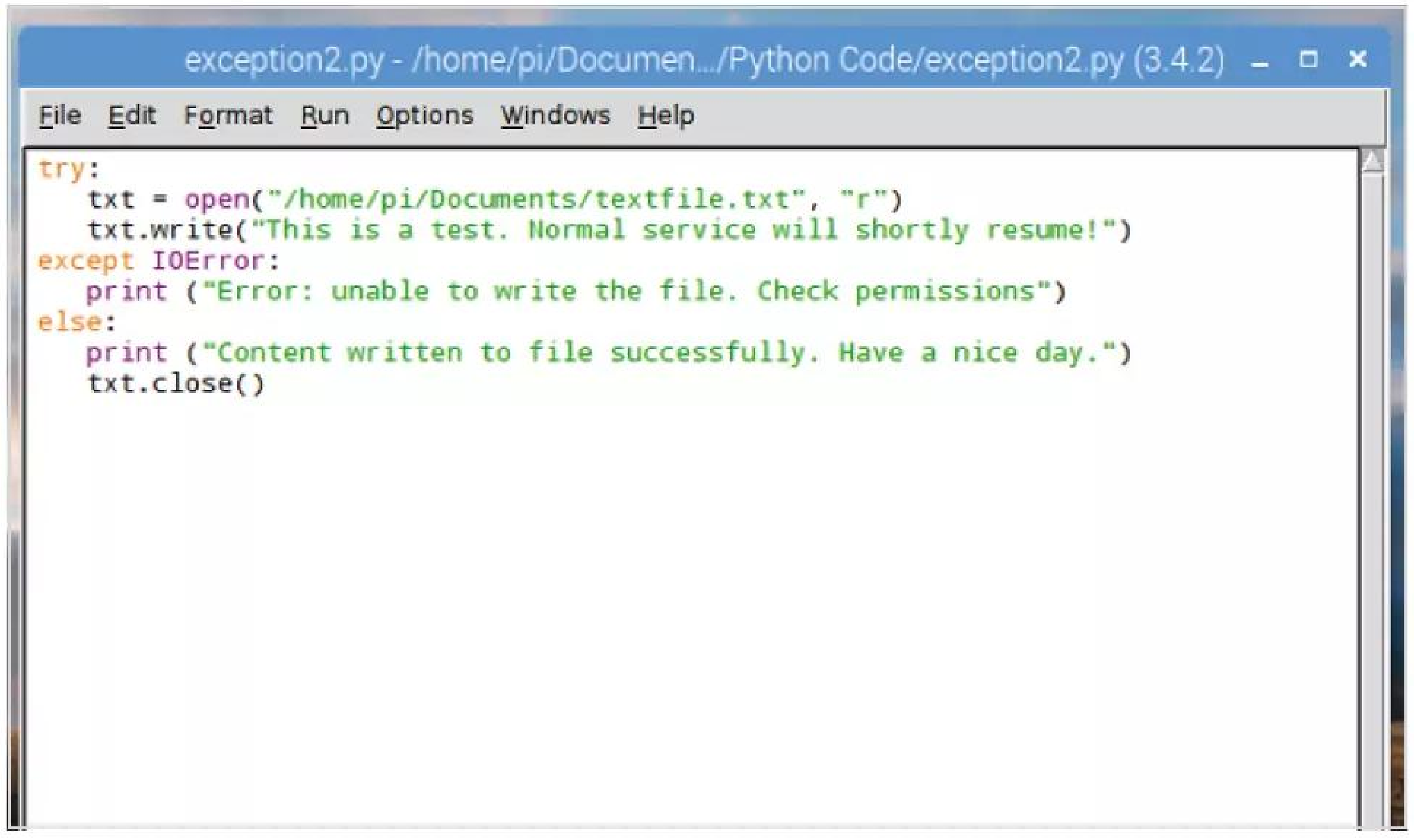

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
exception2.py - /home/pi/Documen.../Python Code/exception2.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Error: unable to write the file. Check permissions
>>>
```

```
try:
    txt = open("/home/pi/Documents/textfile.txt", "r")
    txt.write("This is a test. Normal service will shortly resume!")
except IOError:
    print ("Error: unable to write the file. Check permissions")
else:
    print ("Content written to file successfully. Have a nice day.")
    txt.close()
```

STEP 7

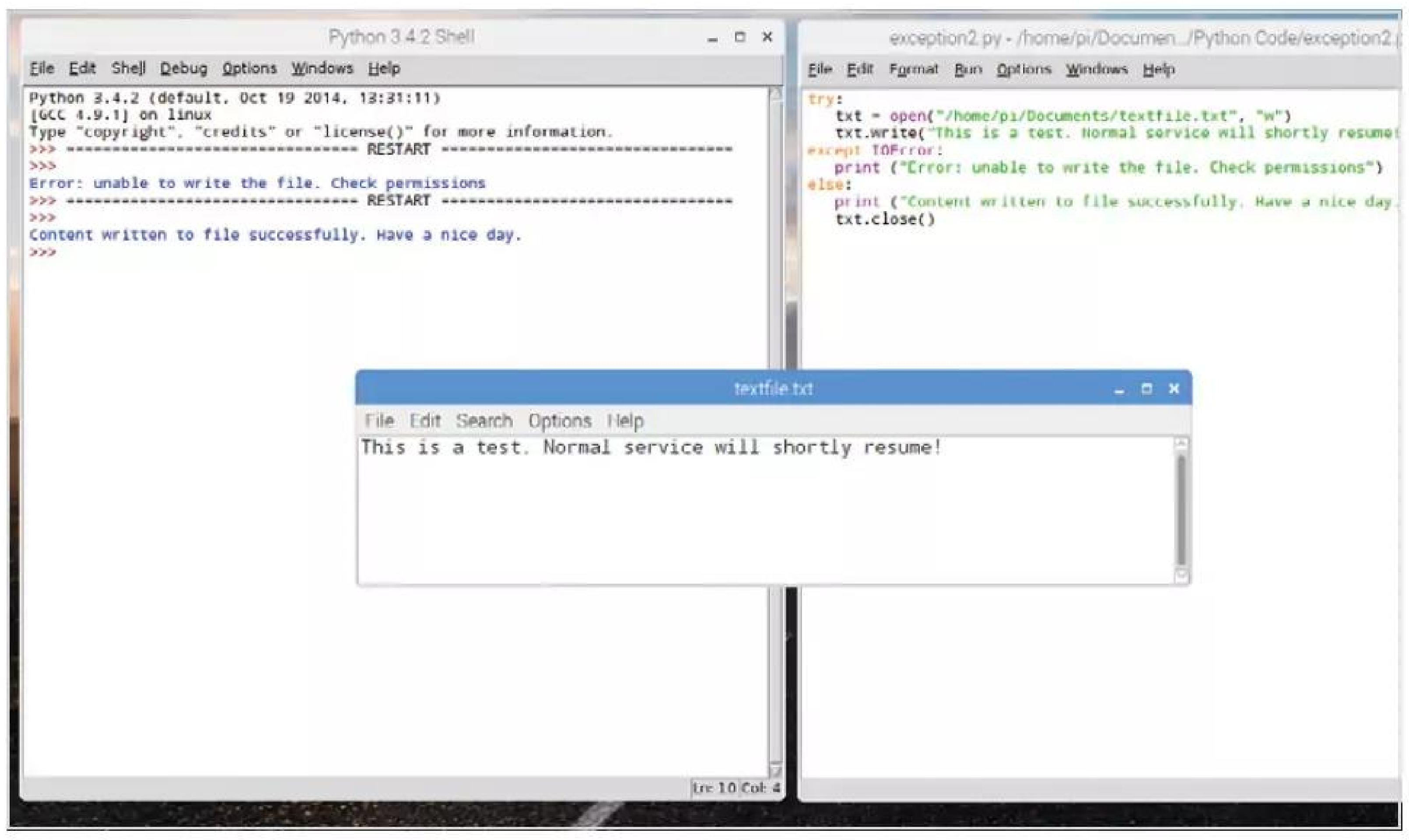
Obviously this won't work due to the file textfile.txt being opened as read only (the "r" part). So in this case rather than Python telling you that you're doing something wrong, you've created an exception using the IOError class informing the user that the permissions are incorrect.


exception2.py - /home/pi/Documen.../Python Code/exception2.py (3.4.2)
File Edit Format Run Options Windows Help

```
try:
    txt = open("/home/pi/Documents/textfile.txt", "r")
    txt.write("This is a test. Normal service will shortly resume!")
except IOError:
    print ("Error: unable to write the file. Check permissions")
else:
    print ("Content written to file successfully. Have a nice day.")
    txt.close()
```

STEP 8

Naturally, you can quickly fix the issue by changing the "r" read only instance with a "w" for write. This, as you already know, will create the file and write the content then commit the changes to the file. The end result will report a different set of circumstances, in this case, a successful execution of the code.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
exception2.py - /home/pi/Documen.../Python Code/exception2.
File Edit Format Run Options Windows Help
textfile.txt
File Edit Search Options Help
Ln: 10 Col: 4

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Error: unable to write the file. Check permissions
>>> ================================ RESTART ================================
>>>
Content written to file successfully. Have a nice day.
>>>
```

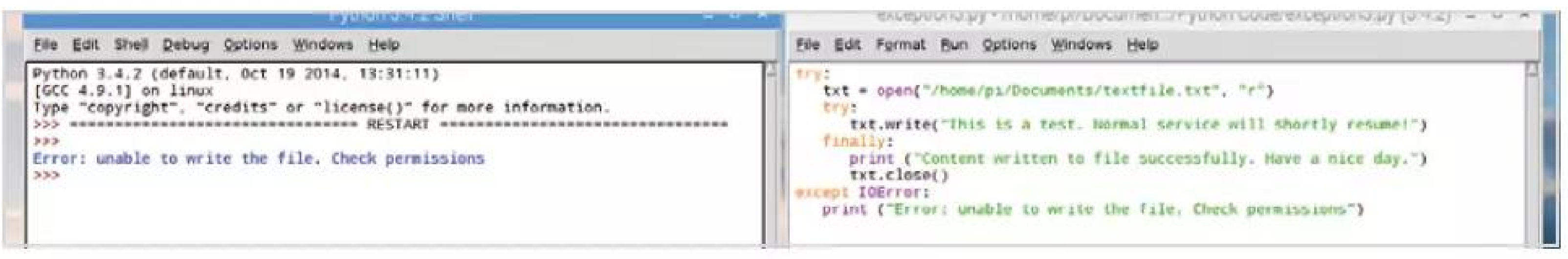
```
try:
    txt = open("/home/pi/Documents/textfile.txt", "w")
    txt.write("This is a test. Normal service will shortly resume
except IOError:
    print ("Error: unable to write the file. Check permissions")
else:
    print ("Content written to file successfully. Have a nice day.
    txt.close()
```

```
This is a test. Normal service will shortly resume!
```

STEP 9

You can also use a finally: block, which works in a similar fashion but you can't use else with it. To use our example from Step 6:

```
try:
 txt = open("/home/pi/Documents/textfile.txt", "r")
 try:
  txt.write("This is a test. Normal service will
shortly resume!")
 finally:
  print ("Content written to file successfully.
Have a nice day.")
  txt.close()
except IOError:
 print ("Error: unable to write the file. Check
permissions")
```

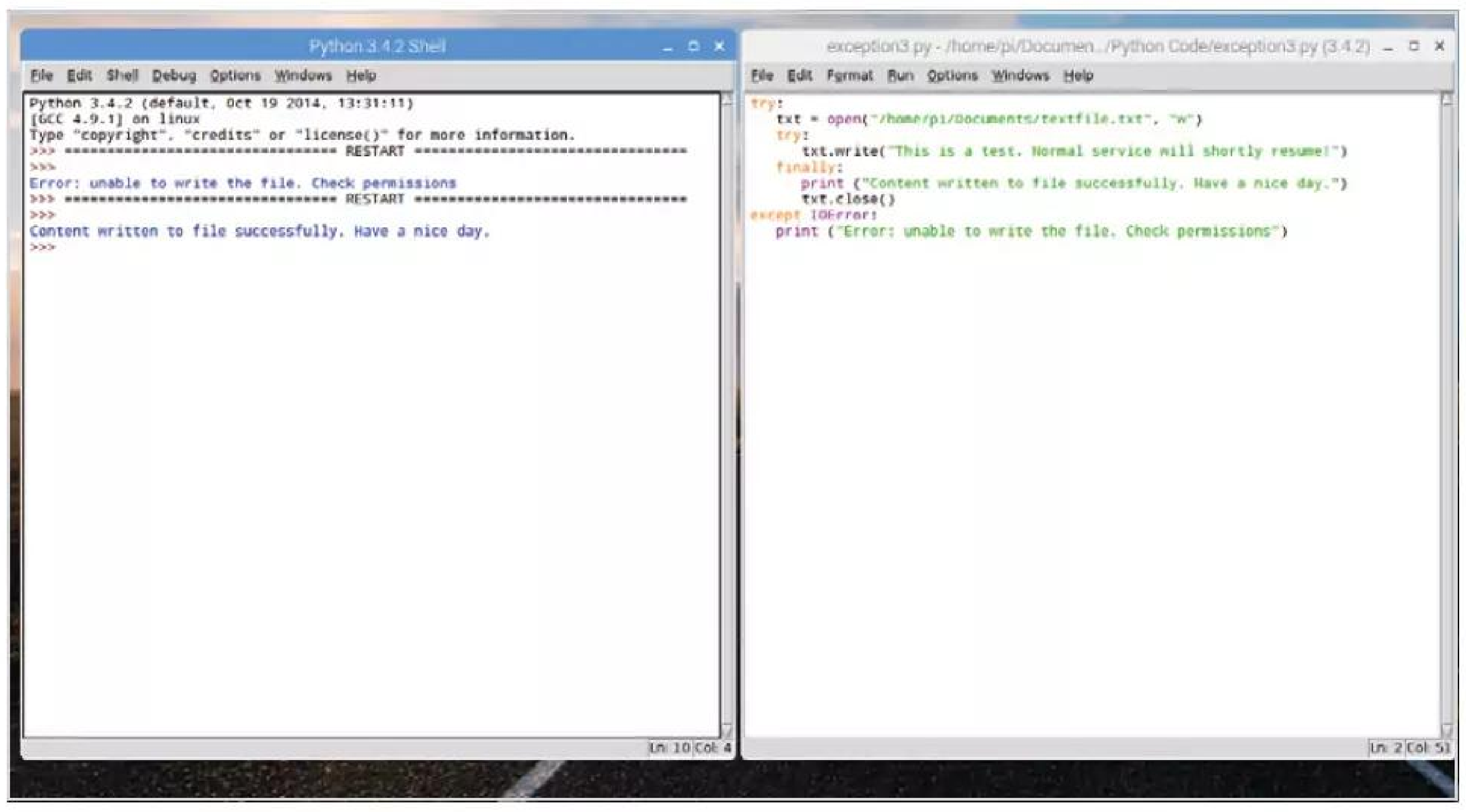

File Edit Shell Debug Options Windows Help
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Error: unable to write the file. Check permissions
>>>
```

```
try:
    txt = open("/home/pi/Documents/textfile.txt", "r")
    try:
        txt.write("This is a test. Normal service will shortly resume!")
    finally:
        print ("Content written to file successfully. Have a nice day.")
        txt.close()
except IOError:
    print ("Error: unable to write the file. Check permissions")
```

STEP 10

As before an error will occur as you've used the "r" read-only permission. If you change it to a "w", then the code will execute without the error being displayed in the IDLE Shell. Needless to say, it can be a tricky getting the exception code right the first time. Practise though, and you will get the hang of it.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
exception3.py - /home/pi/Documen.../Python Code/exception3.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 10 Col: 4
Ln: 2 Col: 51

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Error: unable to write the file. Check permissions
>>> ================================ RESTART ================================
>>>
Content written to file successfully. Have a nice day.
>>>
```

```
try:
    txt = open("/home/pi/Documents/textfile.txt", "w")
    try:
        txt.write("This is a test. Normal service will shortly resume!")
    finally:
        print ("Content written to file successfully. Have a nice day.")
        txt.close()
except IOError:
    print ("Error: unable to write the file. Check permissions")
```

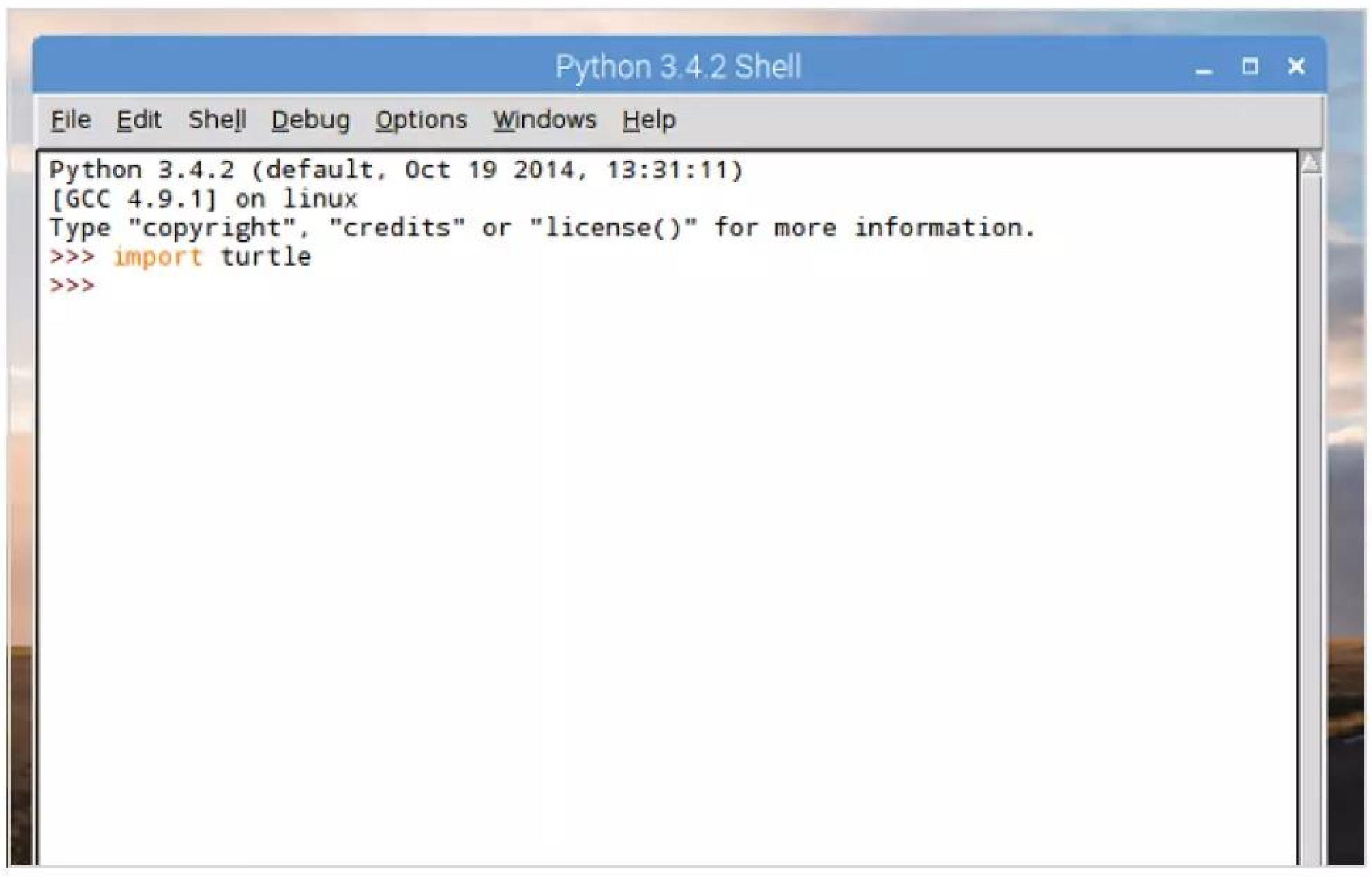
Python Graphics

While dealing with text on the screen, either as a game or in a program, is great, there will come a time when a bit of graphical representation wouldn't go amiss. Python 3 has numerous ways in which to include graphics and they're surprisingly powerful too.

GOING GRAPHICAL

You can draw simple graphics, lines, squares and so on, or you can use one of the many Python modules available, to bring out some spectacular effects.

STEP 1 One of the best graphical modules to begin learning Python graphics is Turtle. The Turtle module is, as the name suggests, based on the turtle robots used in many schools, that can be programmed to draw something on a large piece of paper on the floor. The Turtle module can be imported with: `import turtle`.

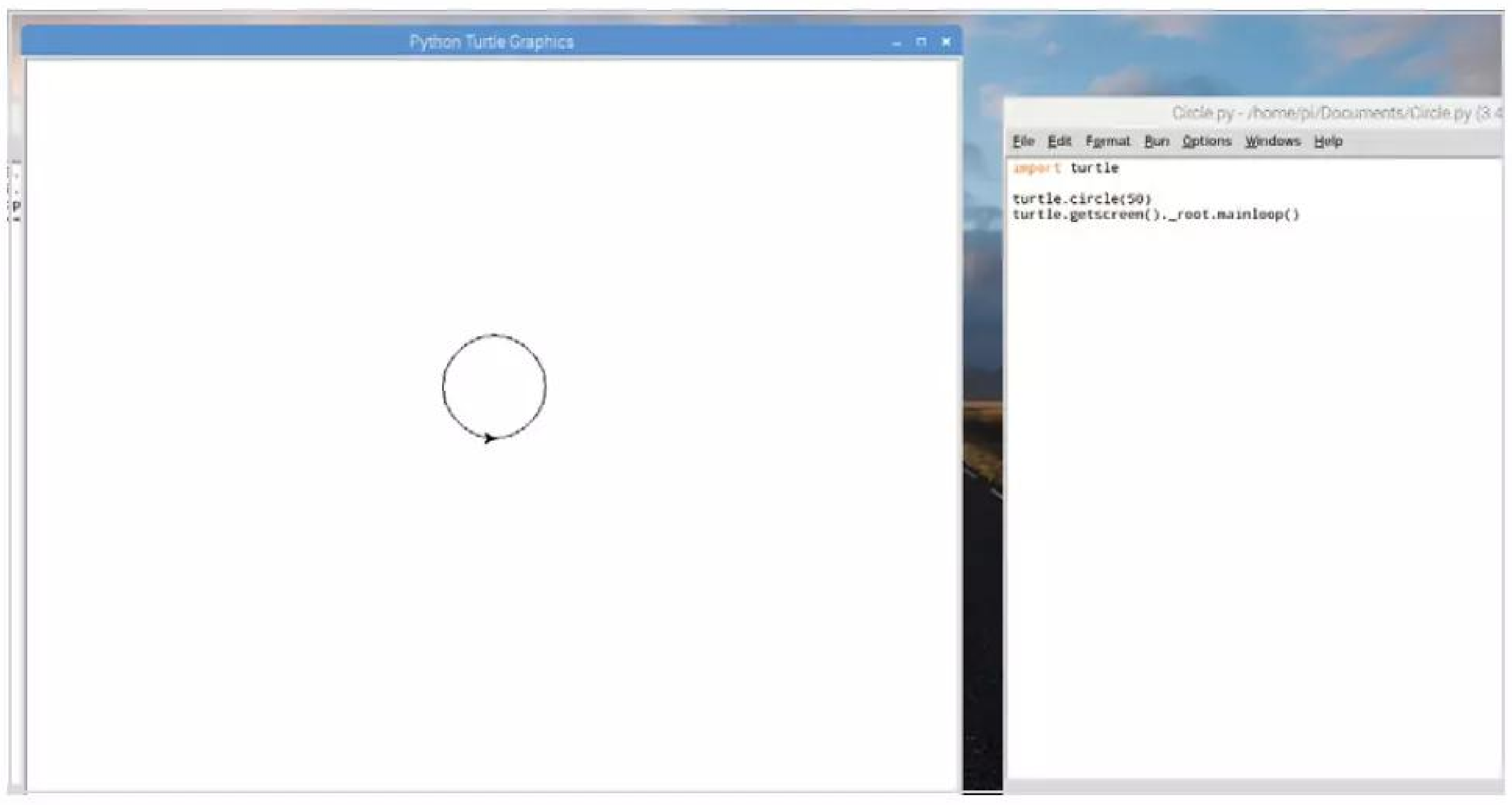

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import turtle
>>>
```

STEP 2 Let's begin by drawing a simple circle. Start a New File, then enter the following code:

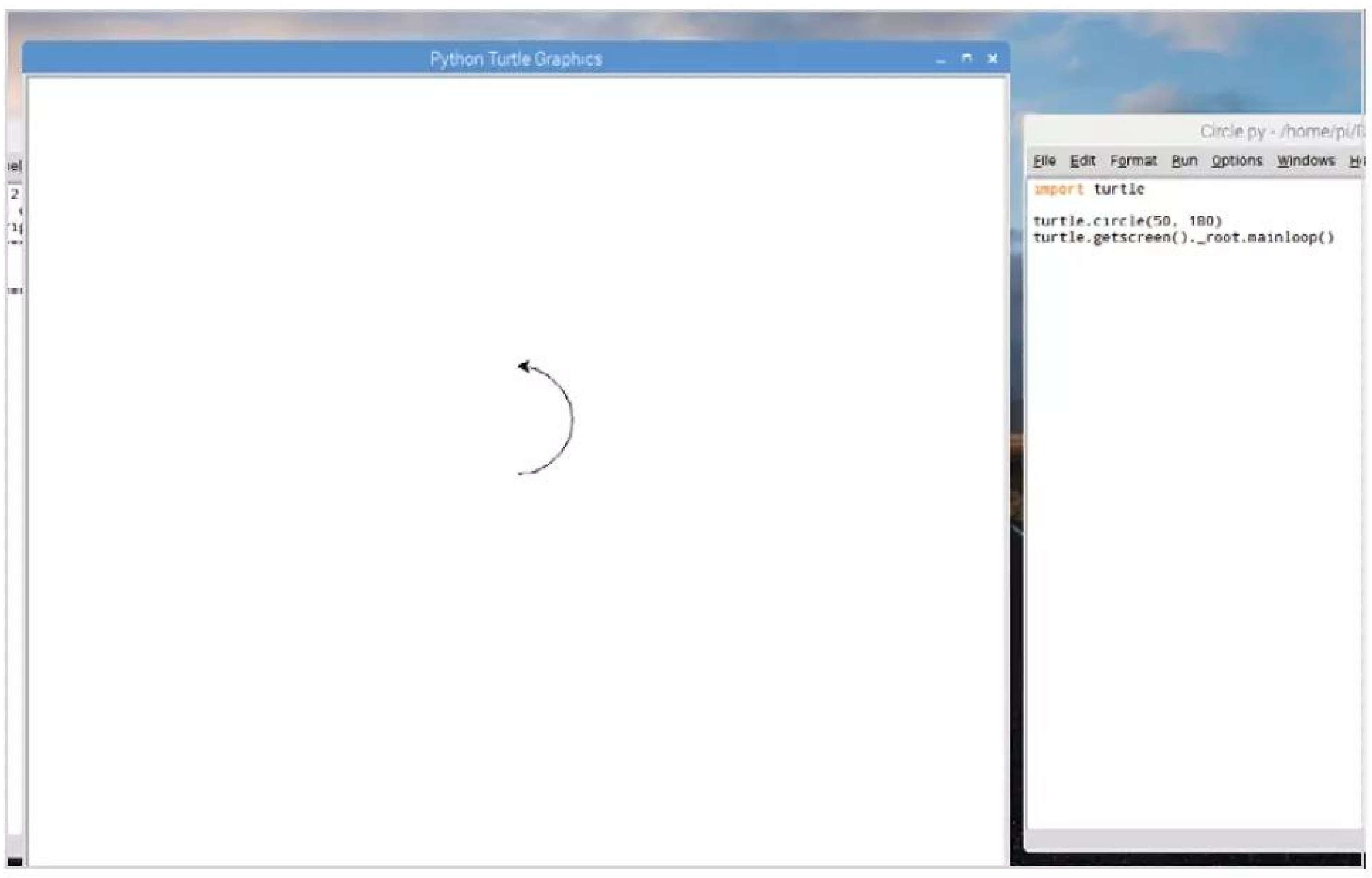
```
import turtle

turtle.circle(50)
turtle.getscreen()._root.mainloop()
```

As usual press F5 to save the code and execute it. A new window will now open up and the 'Turtle' will draw a circle.


Python Turtle Graphics
Circle.py - /home/pi/Documents/Circle.py (3.4
File Edit Format Run Options Windows Help

```
import turtle

turtle.circle(50)
turtle.getscreen()._root.mainloop()
```

STEP 3 The command turtle.circle(50) is what draws the circle on the screen, with 50 being the size. You can play around with the sizes if you like, going up to 100, 150 and beyond; you can draw an arc by entering: `turtle.circle(50, 180)`, where the size is 50, but you're telling Python to only draw 180° of the circle.

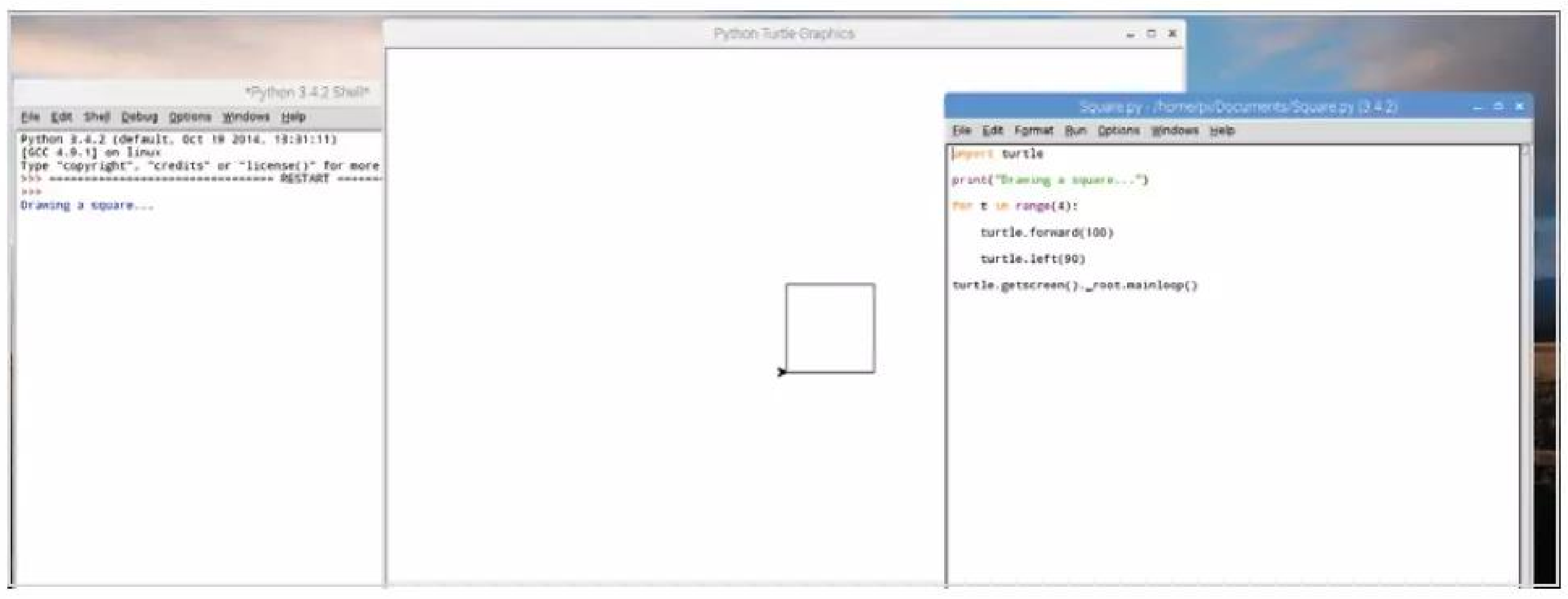

Python Turtle Graphics
Circle.py - /home/pi/
File Edit Format Run Options Windows H

```
import turtle

turtle.circle(50, 180)
turtle.getscreen()._root.mainloop()
```

STEP 4 The last part of the circle code tells Python to keep the window where the drawing is taking place to remain open, so the user can click to close it. Now, let's make a square:

```
import turtle

print("Drawing a square…")

for t in range(4):
   turtle.forward(100)
   turtle.left(90)
turtle.getscreen()._root.mainloop()
```

You can see that we've inserted a loop to draw the sides of the square.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python Turtle Graphics
Square.py - /home/pi/Documents/Square.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more
>>> ================================ RESTART ======
>>>
Drawing a square...
```

```
import turtle

print("Drawing a square...")

for t in range(4):

    turtle.forward(100)

    turtle.left(90)

turtle.getscreen()._root.mainloop()
```

STEP 5

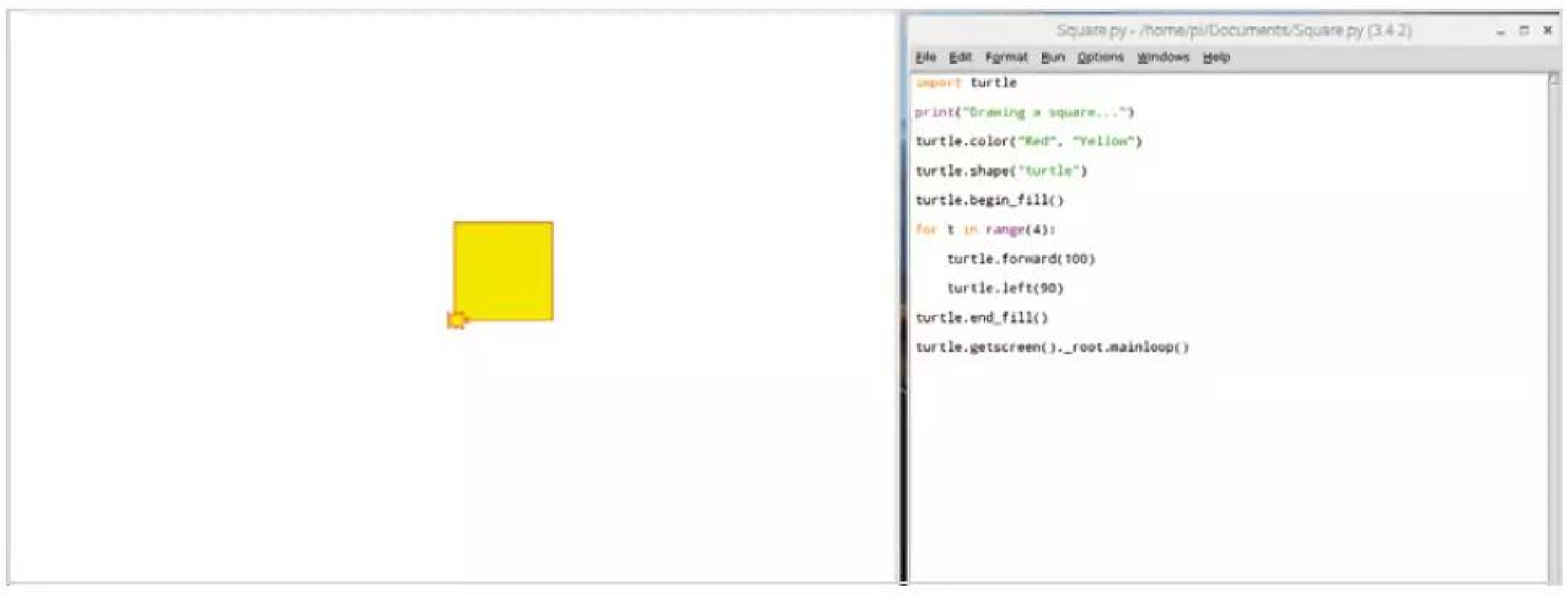
You can add a new line to the square code to add some colour:

```
turtle.color("Red")
```

Then you can even change the character to an actual turtle by entering:

```
turtle.shape("turtle")
```

You can also use the command `turtle.begin_fill()`, and `turtle.end_fill()` to fill in the square with the chosen colours; red outline, yellow fill in this case.

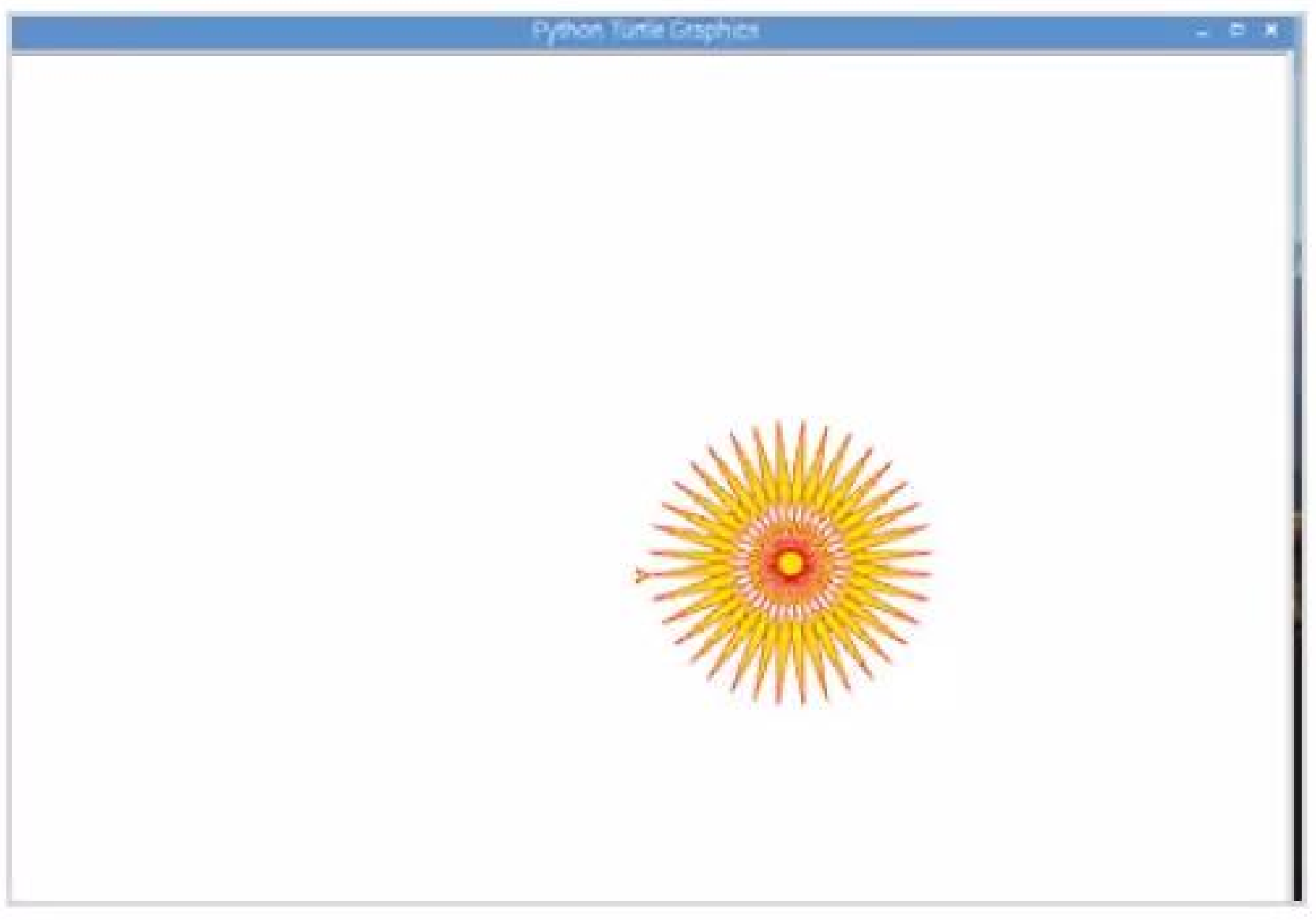

Square.py - /home/pi/Documents/Square.py (3.4.2)
File Edit Format Run Options Windows Help

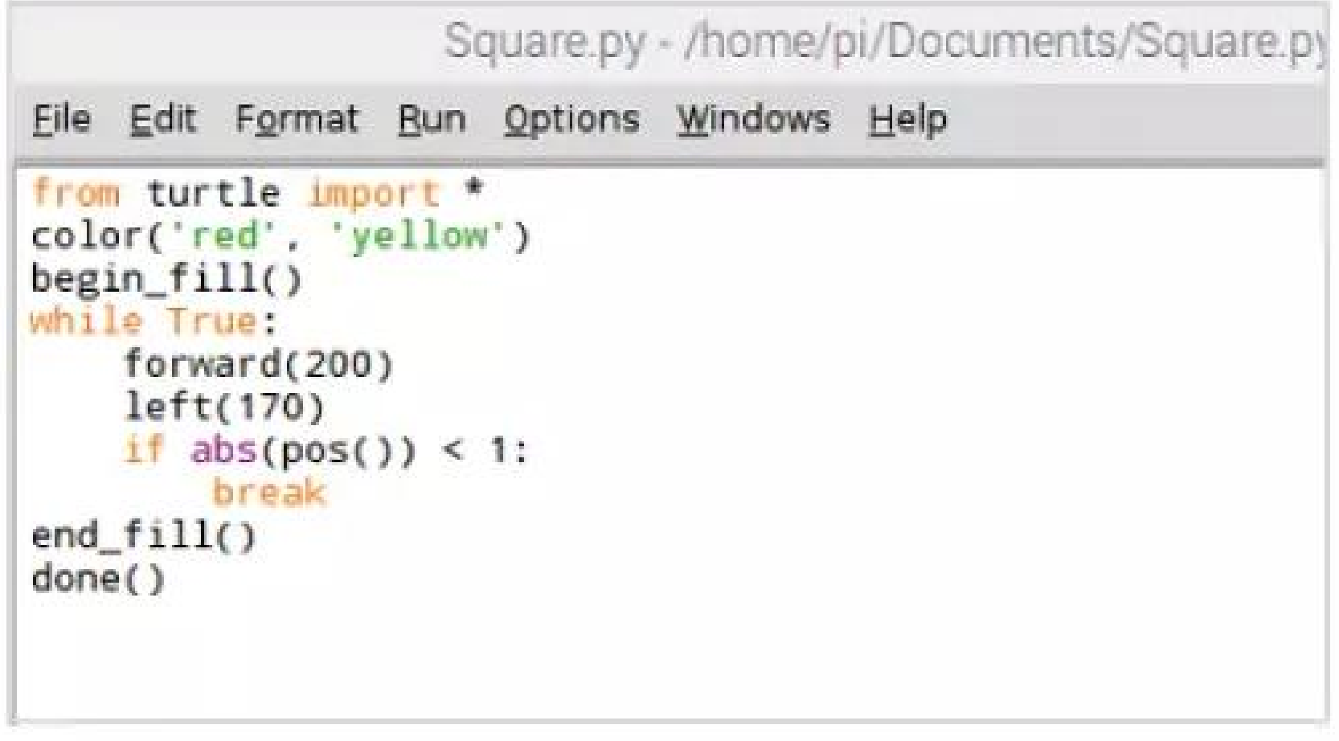
```
import turtle
print("Drawing a square...")
turtle.color("Red", "Yellow")
turtle.shape("turtle")
turtle.begin_fill()
for t in range(4):
    turtle.forward(100)
    turtle.left(90)
turtle.end_fill()
turtle.getscreen()._root.mainloop()
```

STEP 6

You can see that the Turtle module can draw out some pretty good shapes and become a little more complex as you begin to master the way it works. Enter this example:

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
  forward(200)
  left(170)
  if abs(pos()) < 1:
    break
end_fill()
done()
```


Python Turtle Graphics

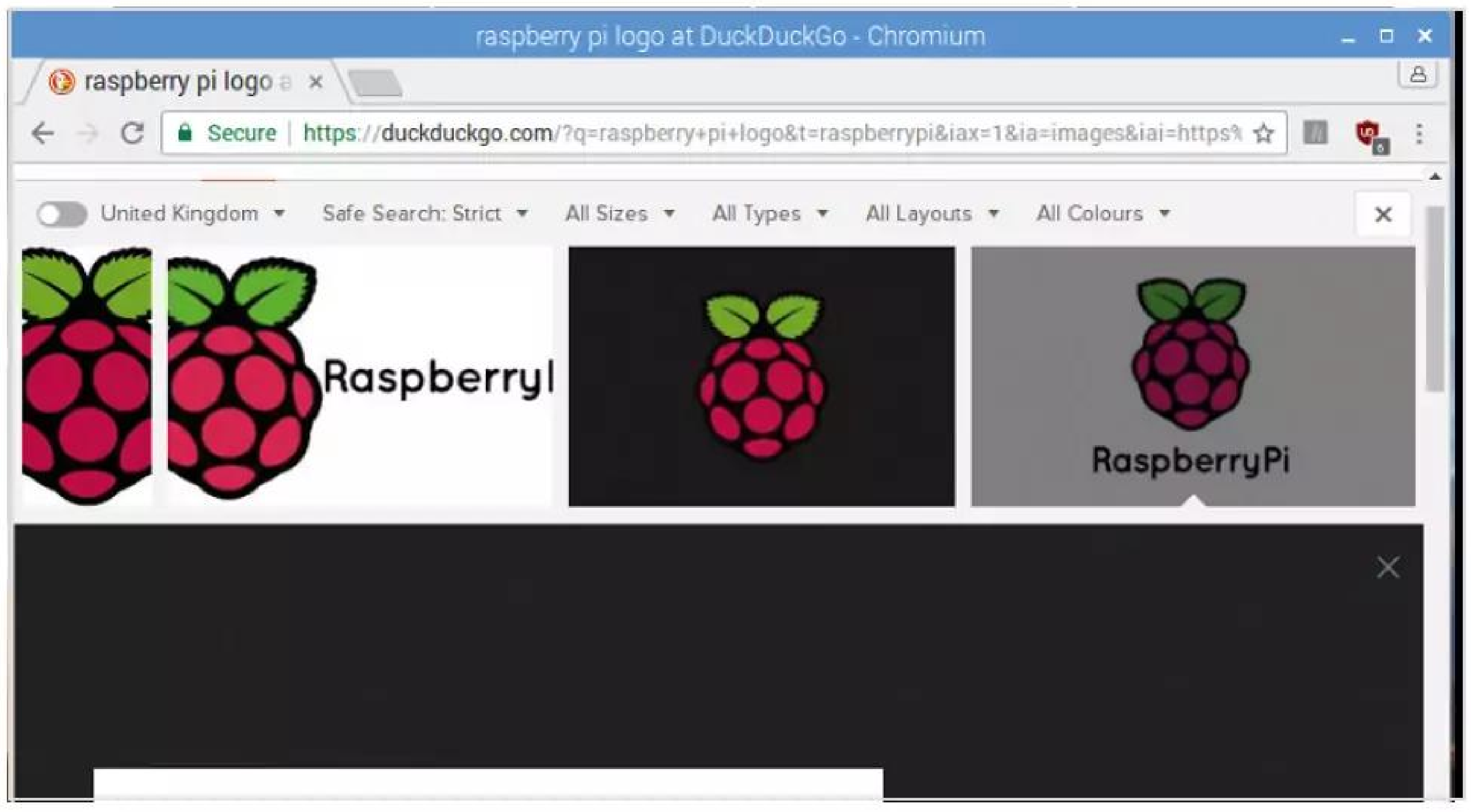

Square.py - /home/pi/Documents/Square.py
File Edit Format Run Options Windows Help

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

It's a different method, but very effective.

STEP 7

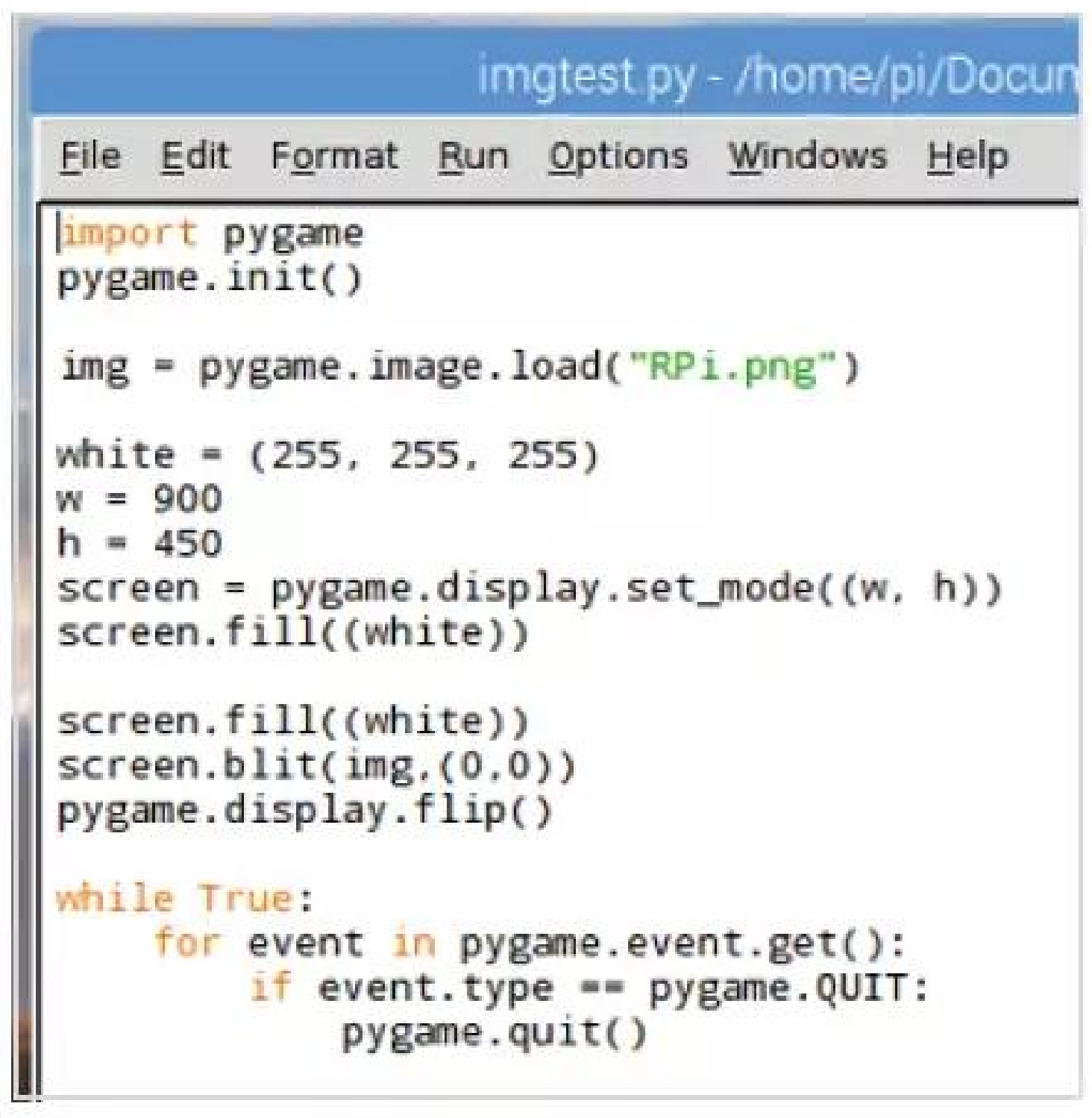
Another way in which you can display graphics is by using the Pygame module. There are numerous ways in which pygame can help you output graphics to the screen but for now let's look at displaying a predefined image. Start by opening a browser and finding an image, then save it to the folder where you save your Python code.


raspberry pi logo at DuckDuckGo - Chromium
raspberry pi logo
Secure
https://duckduckgo.com/?q=raspberry+pi+logo&t=raspberrypi&iax=1&ia=images&iai=https%
United Kingdom
Safe Search: Strict
All Sizes
All Types
All Layouts
All Colours
Raspberry
RaspberryPi

STEP 8

Now let's get the code by importing the pygame module:

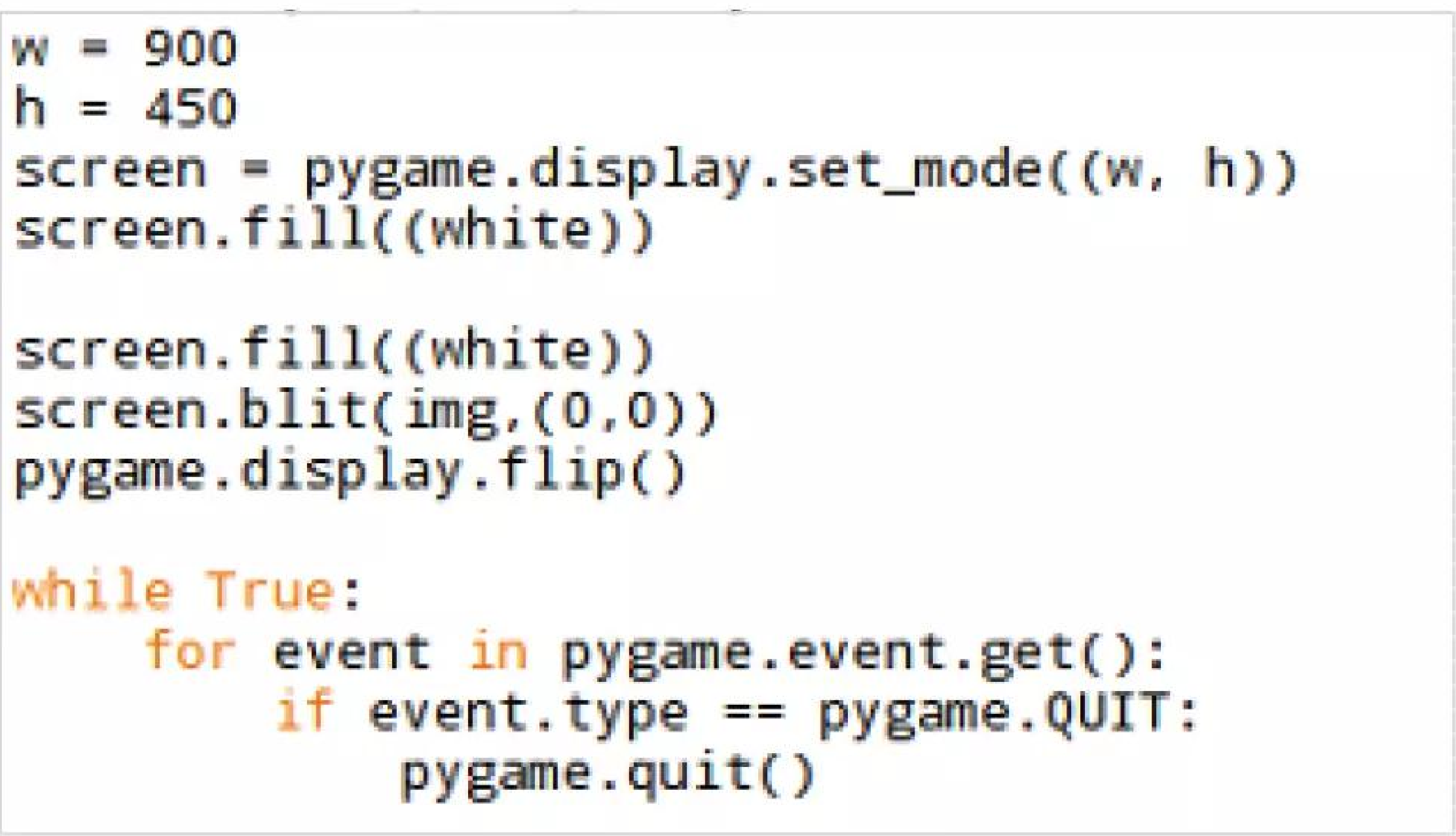
```
import pygame
pygame.init()

img = pygame.image.load("RPi.png")

white = (255, 255, 255)
w = 900
h = 450
screen = pygame.display.
set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
 for event in pygame.event.get():
   if event.type == pygame.QUIT:
     pygame.quit()
```


imgtest.py - /home/pi/Docun
File Edit Format Run Options Windows Help

```
import pygame
pygame.init()

img = pygame.image.load("RPi.png")

white = (255, 255, 255)
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0.0))
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
          pygame.quit()
```

STEP 9

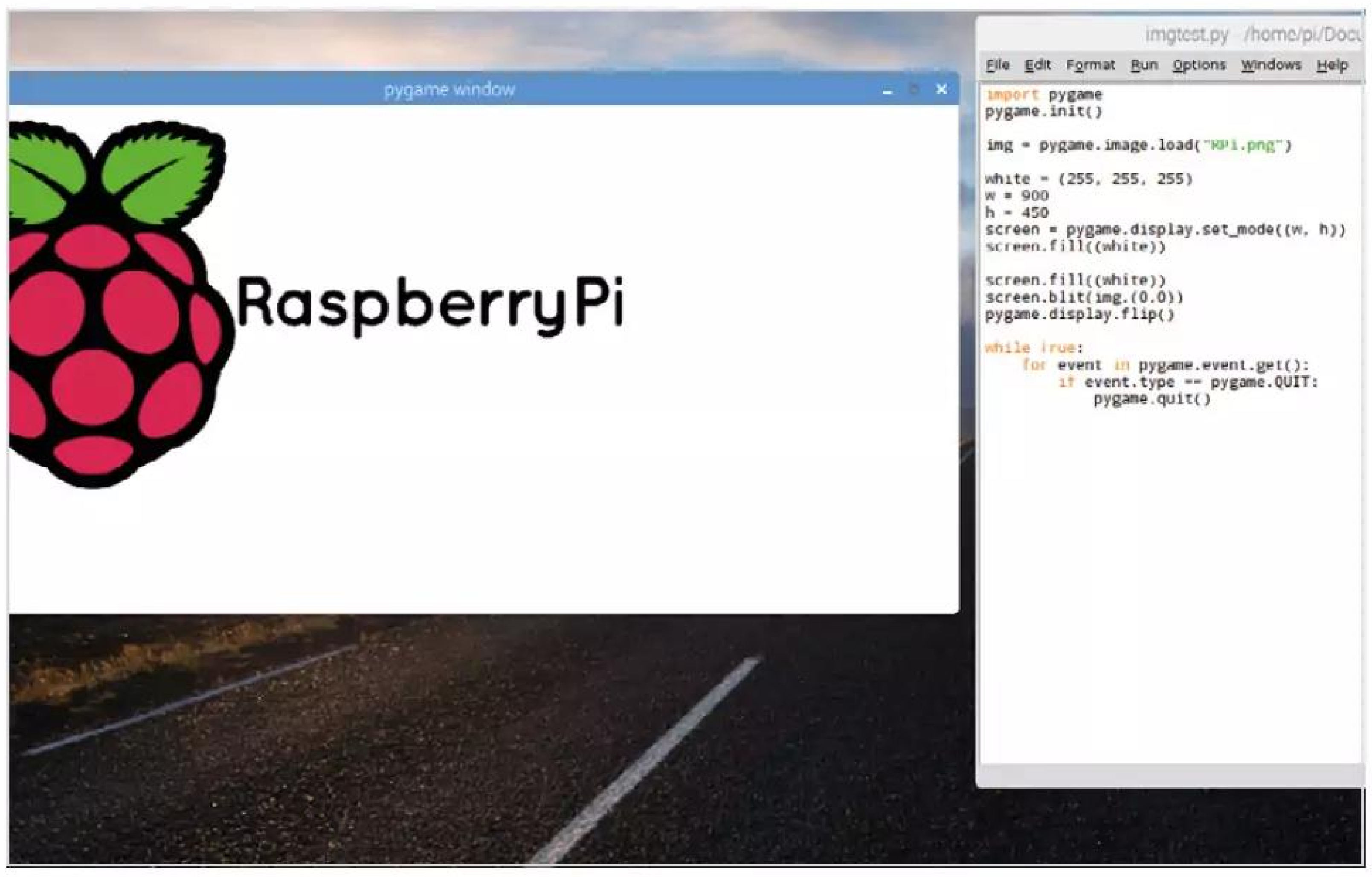
In the previous step you imported pygame, initiated the pygame engine and asked it to import our saved Raspberry Pi logo image, saved as RPi.png. Next you defined the background colour of the window to display the image and the window size as per the actual image dimensions. Finally you have a loop to close the window.



```
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img,(0,0))
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
```

STEP 10

Press F5 to save and execute the code and your image will be displayed in a new window. Have a play around with the colours, sizes and so on and take time to look up the many functions within the pygame module too.


pygame window
RaspberryPi
imgtest.py /home/pi/Docu
File Edit Format Run Options Windows Help

```
import pygame
pygame.init()

img = pygame.image.load("RPi.png")

white = (255, 255, 255)
w = 900
h = 450
screen = pygame.display.set_mode((w, h))
screen.fill((white))

screen.fill((white))
screen.blit(img.(0.0))
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
          pygame.quit()
```

Using Modules

A Python module is simply a Python-created source file which contains the necessary code for classes, functions and global variables. You can bind and reference modules to extend functionality and create even more spectacular Python programs.

Want to see how to improve these modules to add a little something extra to your code? Then read on and learn how they can be used to fashion fantastic code.

Calendar Module

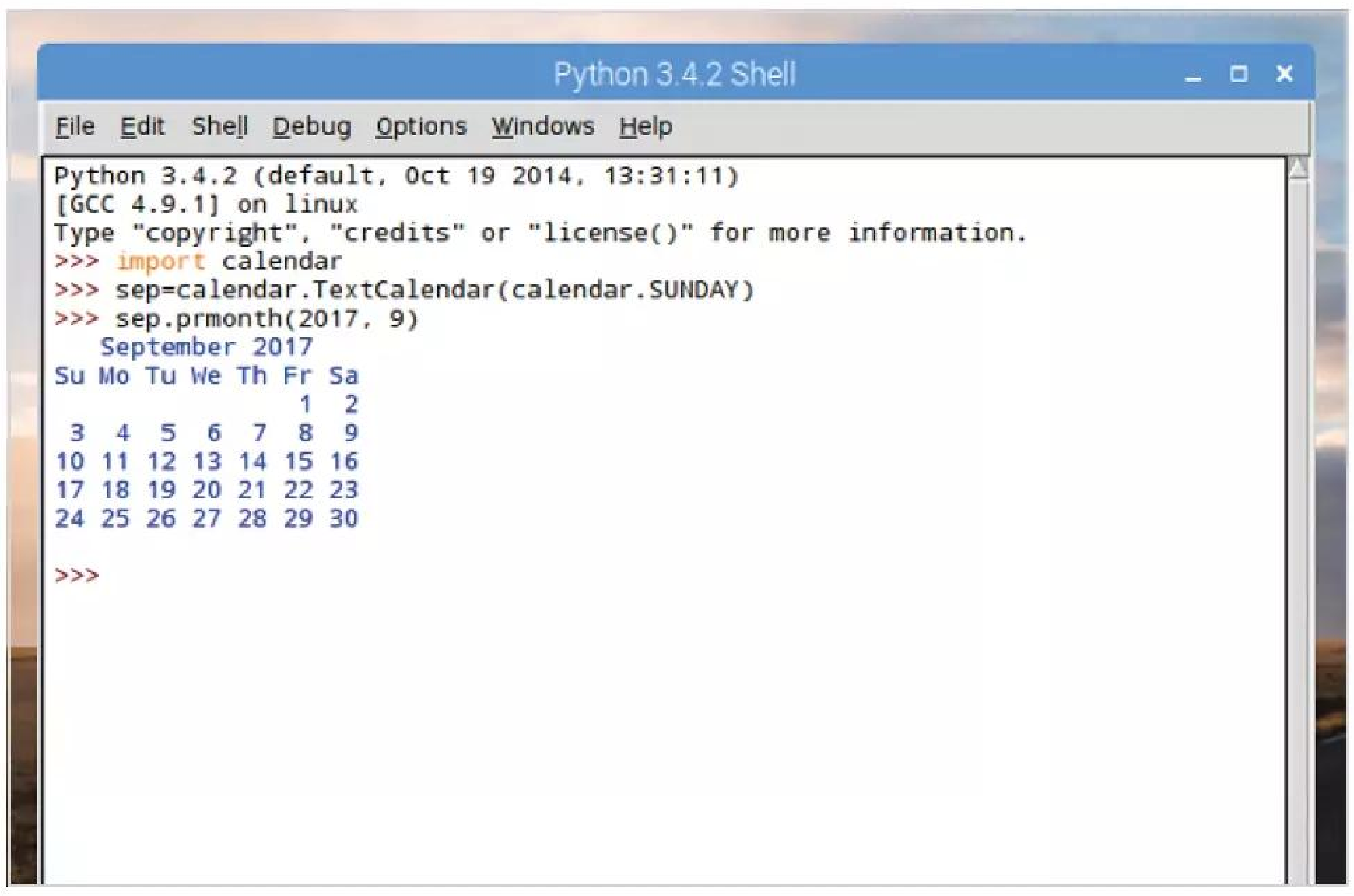
Beyond the time module, the calendar module can produce some interesting results when executed within your code. It does far more than simply display the date in the time module-like format, you can actually call up a wall calendar type display.

WORKING WITH DATES

The calendar module is built into Python 3. However, if for some reason it's not installed you can add it using **pip install calendar** as a Windows administrator or **sudo pip install calendar** for Linux and macOS.

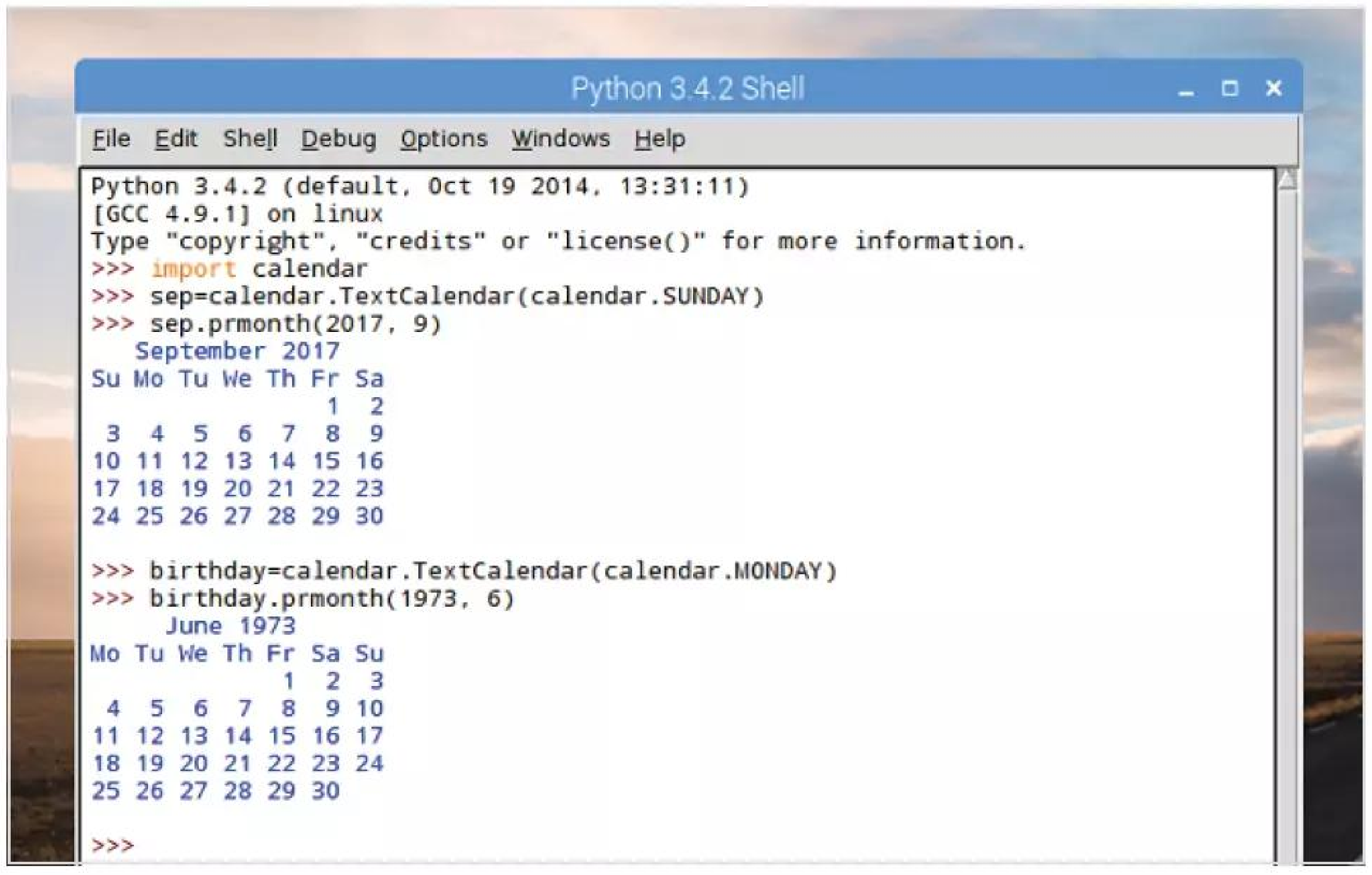
STEP 1 Launch Python 3 and enter: `import calendar` to call up the module and its inherent functions. Once it's loaded into memory, start by entering:

```
sep=calendar.TextCalendar(calendar.SUNDAY)
sep.prmonth(2017, 9)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> sep=calendar.TextCalendar(calendar.SUNDAY)
>>> sep.prmonth(2017, 9)
   September 2017
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

>>>
```

STEP 2 You can see that the days of September 2017 are displayed in a wall calendar fashion. Naturally you can change the 2017, 9 part of the second line to any year and month you want, a birthday for example (1973, 6). The first line configures TextCalendar to start its weeks on a Sunday; you can opt for Monday if you prefer.

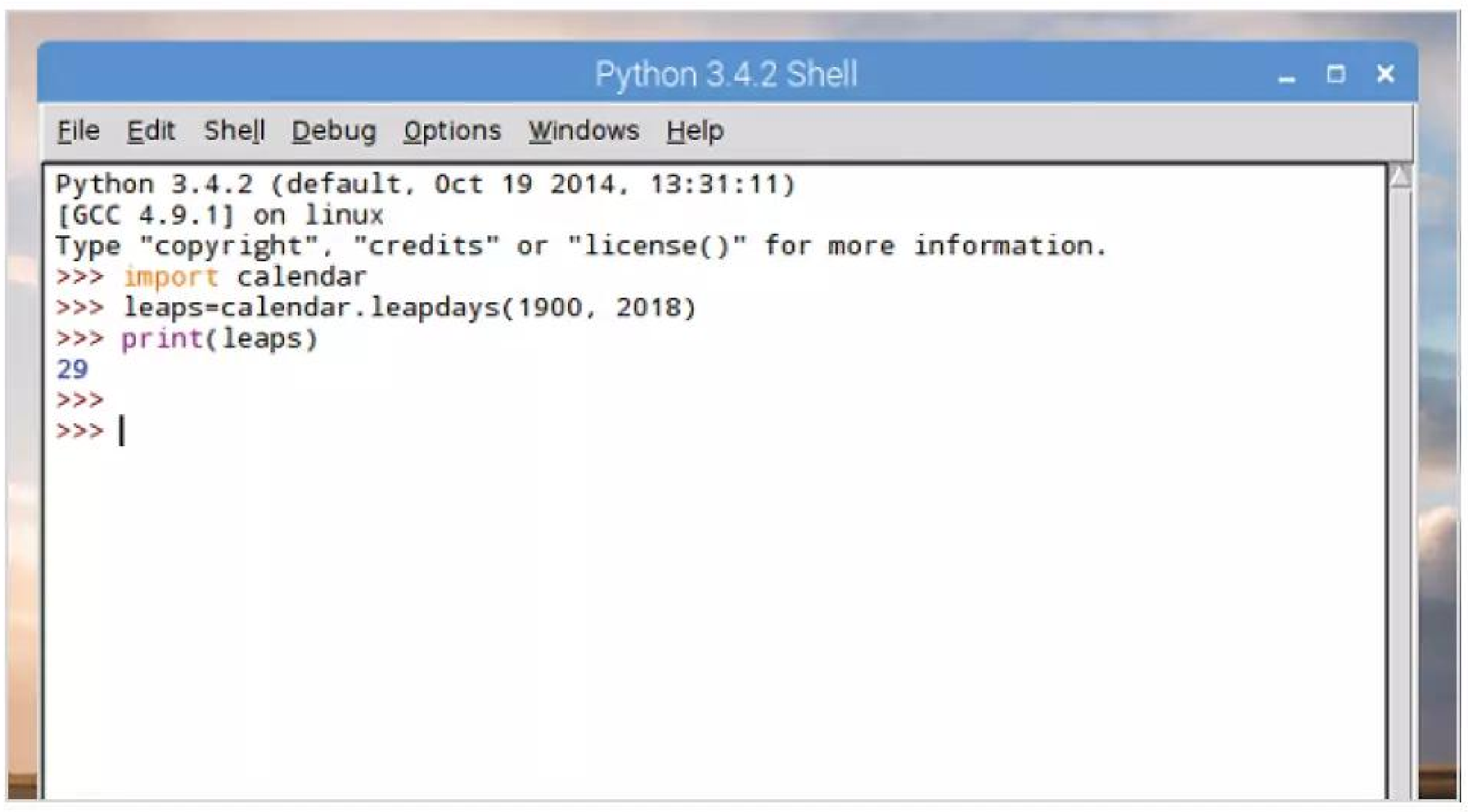

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> sep=calendar.TextCalendar(calendar.SUNDAY)
>>> sep.prmonth(2017, 9)
   September 2017
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

>>> birthday=calendar.TextCalendar(calendar.MONDAY)
>>> birthday.prmonth(1973, 6)
     June 1973
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

>>>
```

STEP 3 There are numerous functions within the calendar module that may be of interest to you when forming your own code. For example, you can display the number of leap years between two specific years:

```
leaps=calendar.leapdays(1900, 2018)
print(leaps)
```

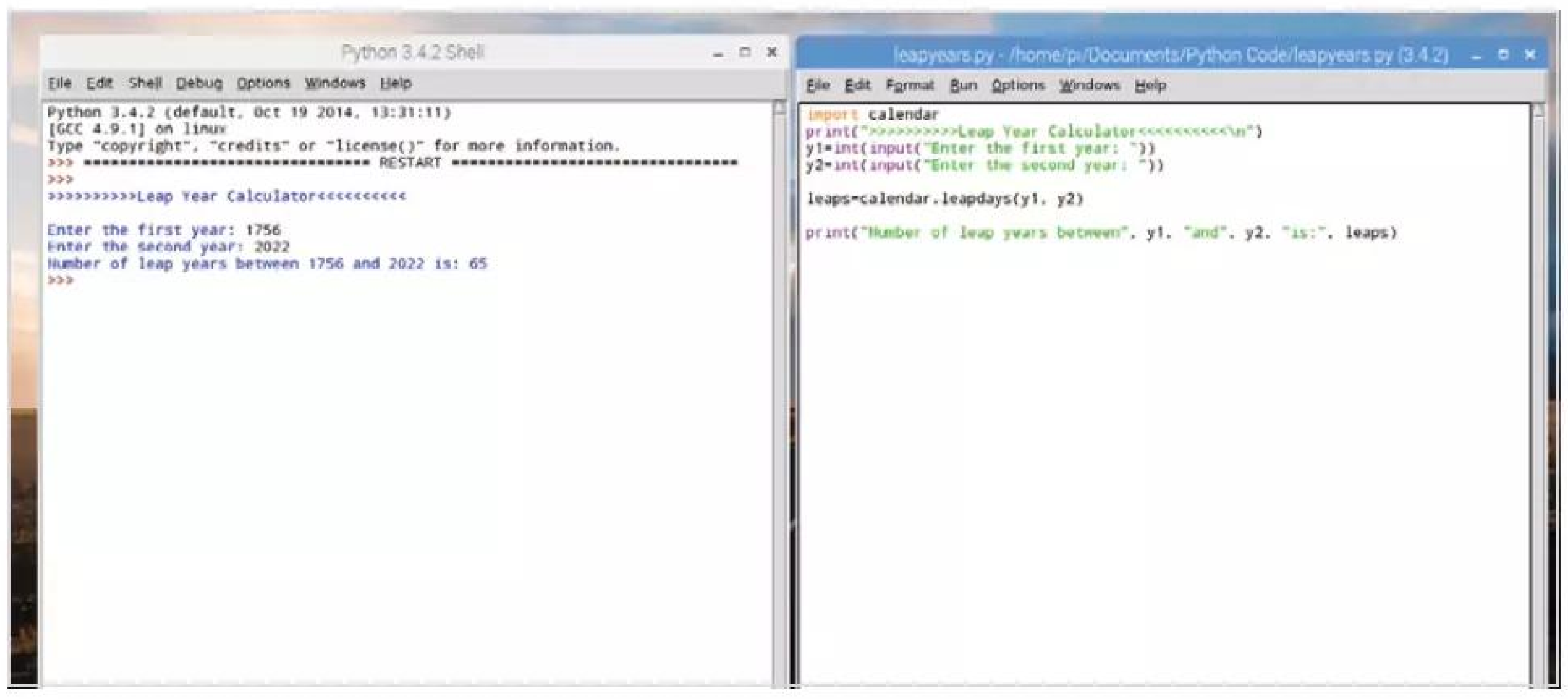
The result is 29, starting from 1904 onward.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> leaps=calendar.leapdays(1900, 2018)
>>> print(leaps)
29
>>>
>>> |
```

STEP 4 You could even fashion that particular example into a piece of working, user interactive Python code:

```
import calendar
print(“>>>>>>>>>>Leap Year
Calculator<<<<<<<<<<\n”)y1=int(input(“Enter the
first year: ”))
y2=int(input(“Enter the second year: ”))
leaps=calendar.leapdays(y1, y2)
print(“Number of leap years between”, y1, “and”,
y2, “is:”, leaps)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
leapyears.py - /home/pi/Documents/Python Code/leapyears.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>>>>>>>>>Leap Year Calculator<<<<<<<<<<

Enter the first year: 1756
Enter the second year: 2022
Number of leap years between 1756 and 2022 is: 65
>>>
```

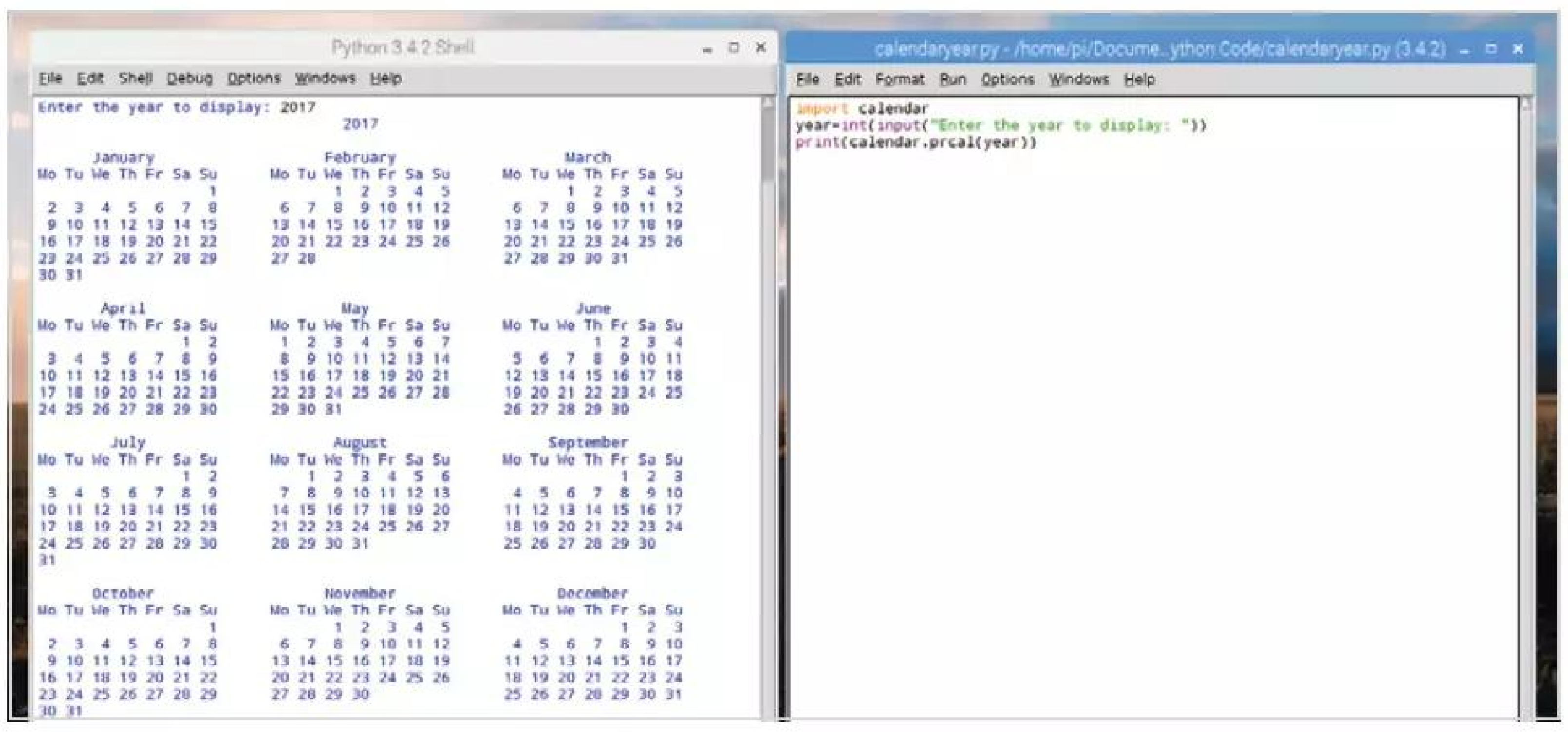
```
import calendar
print(">>>>>>>>>>Leap Year Calculator<<<<<<<<<<\n")
y1=int(input("Enter the first year: "))
y2=int(input("Enter the second year: "))

leaps=calendar.leapdays(y1, y2)

print("Number of leap years between", y1, "and", y2, "is:", leaps)
```

STEP 5

You can also create a program that will display all the days, weeks and months within a given year:

```
import calendar
year=int(input("Enter the year to display: ")
print(calendar.prcal(year))
```

We're sure you'll agree that's quite a handy bit of code to have to hand.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
calendaryear.py - /home/pi/Docume...ython Code/calendaryear.py (3.4.2)
File Edit Format Run Options Windows Help

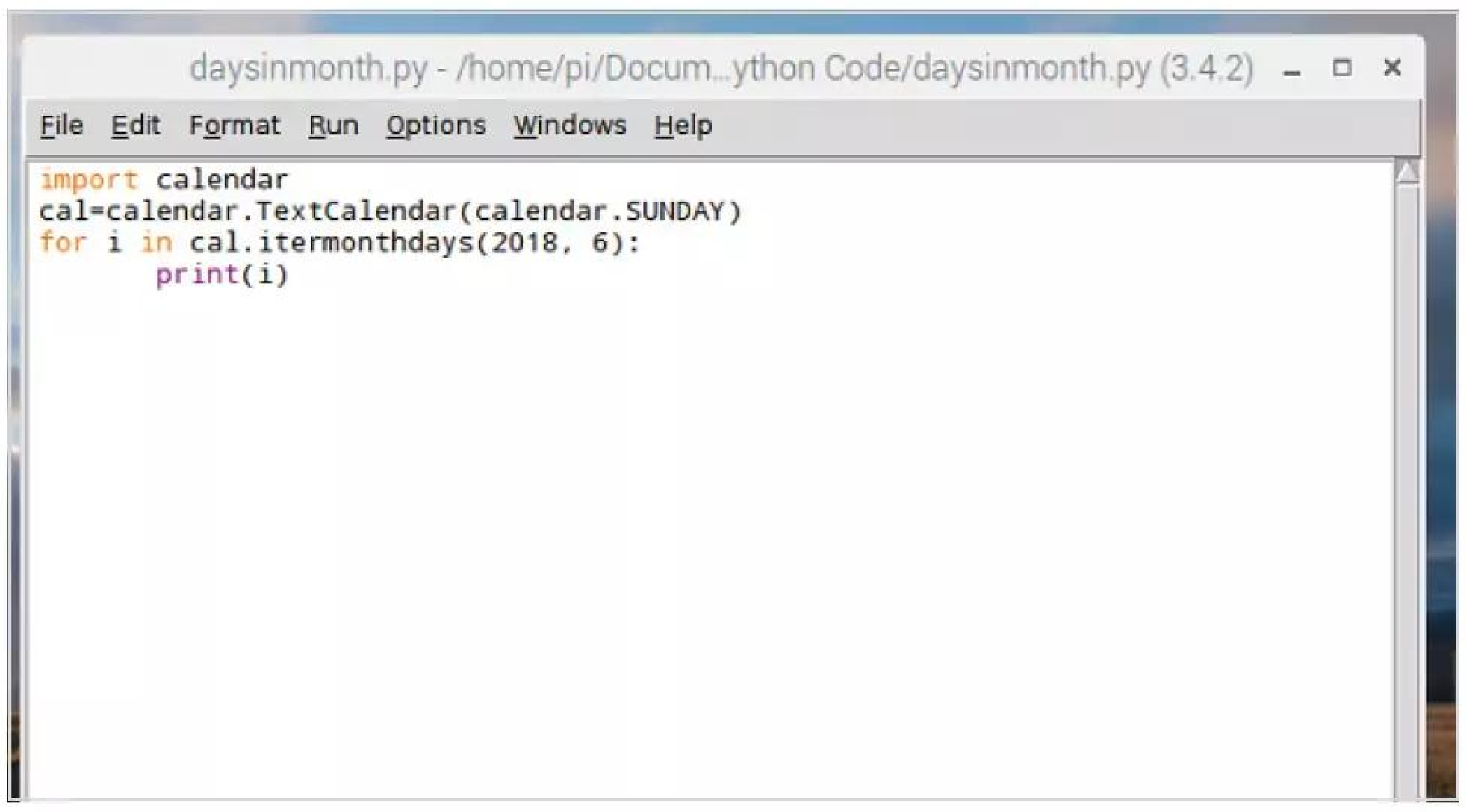
```
Enter the year to display: 2017
                                  2017

      January                   February                   March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1             1  2  3  4  5             1  2  3  4  5
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       6  7  8  9 10 11 12
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      13 14 15 16 17 18 19
16 17 18 19 20 21 22      20 21 22 23 24 25 26      20 21 22 23 24 25 26
23 24 25 26 27 28 29      27 28                     27 28 29 30 31
30 31

       April                      May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2       1  2  3  4  5  6  7                1  2  3  4
 3  4  5  6  7  8  9       8  9 10 11 12 13 14       5  6  7  8  9 10 11
10 11 12 13 14 15 16      15 16 17 18 19 20 21      12 13 14 15 16 17 18
17 18 19 20 21 22 23      22 23 24 25 26 27 28      19 20 21 22 23 24 25
24 25 26 27 28 29 30      29 30 31                  26 27 28 29 30

        July                     August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2          1  2  3  4  5  6                   1  2  3
 3  4  5  6  7  8  9       7  8  9 10 11 12 13       4  5  6  7  8  9 10
10 11 12 13 14 15 16      14 15 16 17 18 19 20      11 12 13 14 15 16 17
17 18 19 20 21 22 23      21 22 23 24 25 26 27      18 19 20 21 22 23 24
24 25 26 27 28 29 30      28 29 30 31               25 26 27 28 29 30
31

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1             1  2  3  4  5                   1  2  3
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       4  5  6  7  8  9 10
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      11 12 13 14 15 16 17
16 17 18 19 20 21 22      20 21 22 23 24 25 26      18 19 20 21 22 23 24
23 24 25 26 27 28 29      27 28 29 30               25 26 27 28 29 30 31
30 31
```

```
import calendar
year=int(input("Enter the year to display: "))
print(calendar.prcal(year))
```

STEP 6

Interestingly we can also list the number of days in a month by using a simple for loop:

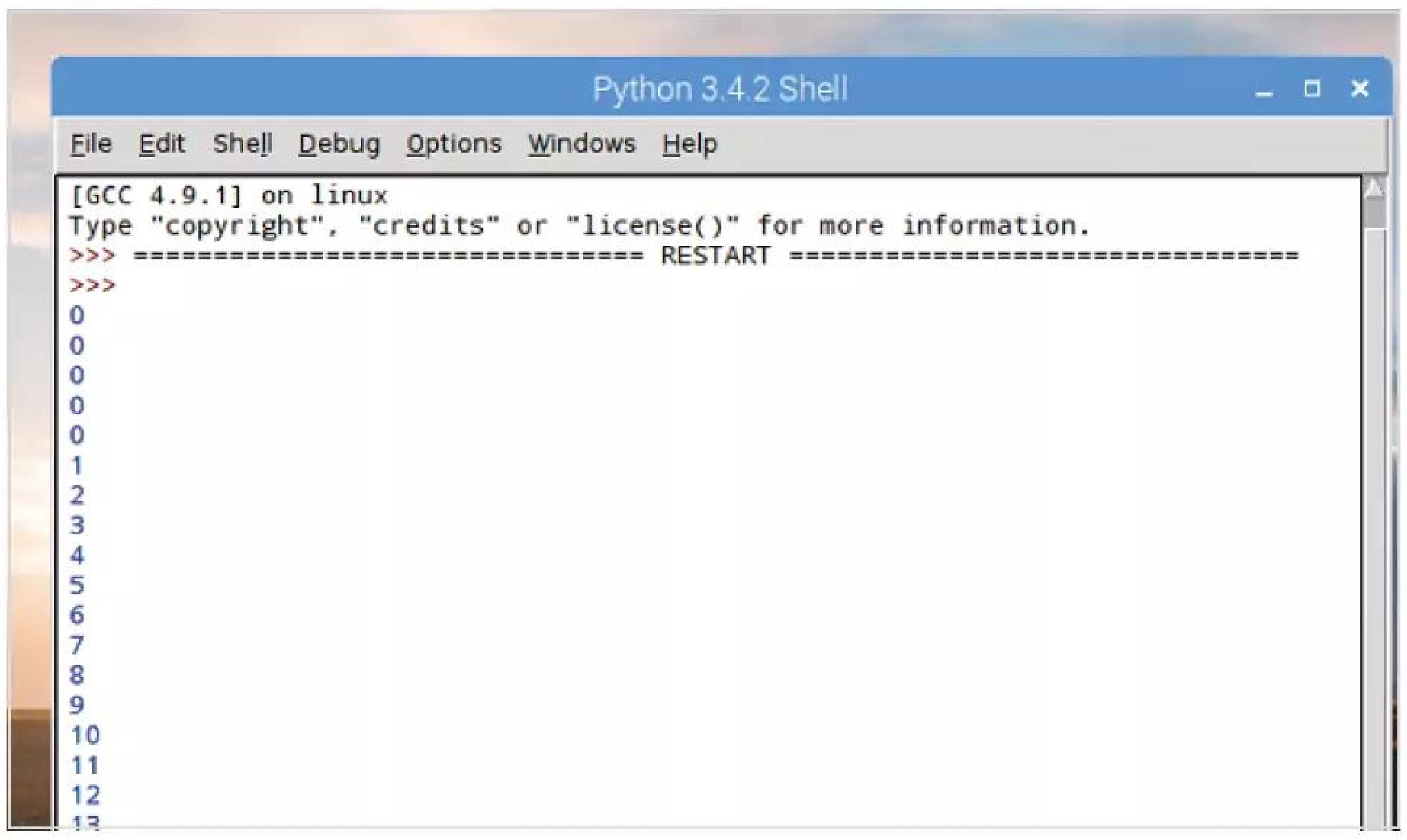
```
import calendar
cal=calendar.TextCalendar(calendar.SUNDAY)
for i in cal.itermonthdays(2018, 6):
   print(i)
```


daysinmonth.py - /home/pi/Docum...ython Code/daysinmonth.py (3.4.2)
File Edit Format Run Options Windows Help

```
import calendar
cal=calendar.TextCalendar(calendar.SUNDAY)
for i in cal.itermonthdays(2018, 6):
        print(i)
```

STEP 7

You can see that code produced some zeros at the beginning, this is due to the starting day of the week, Sunday in this case, and overlapping days from the previous month. So the counting of the days will start on Friday 1st June 2018 and will total 30 as the output correctly displays.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

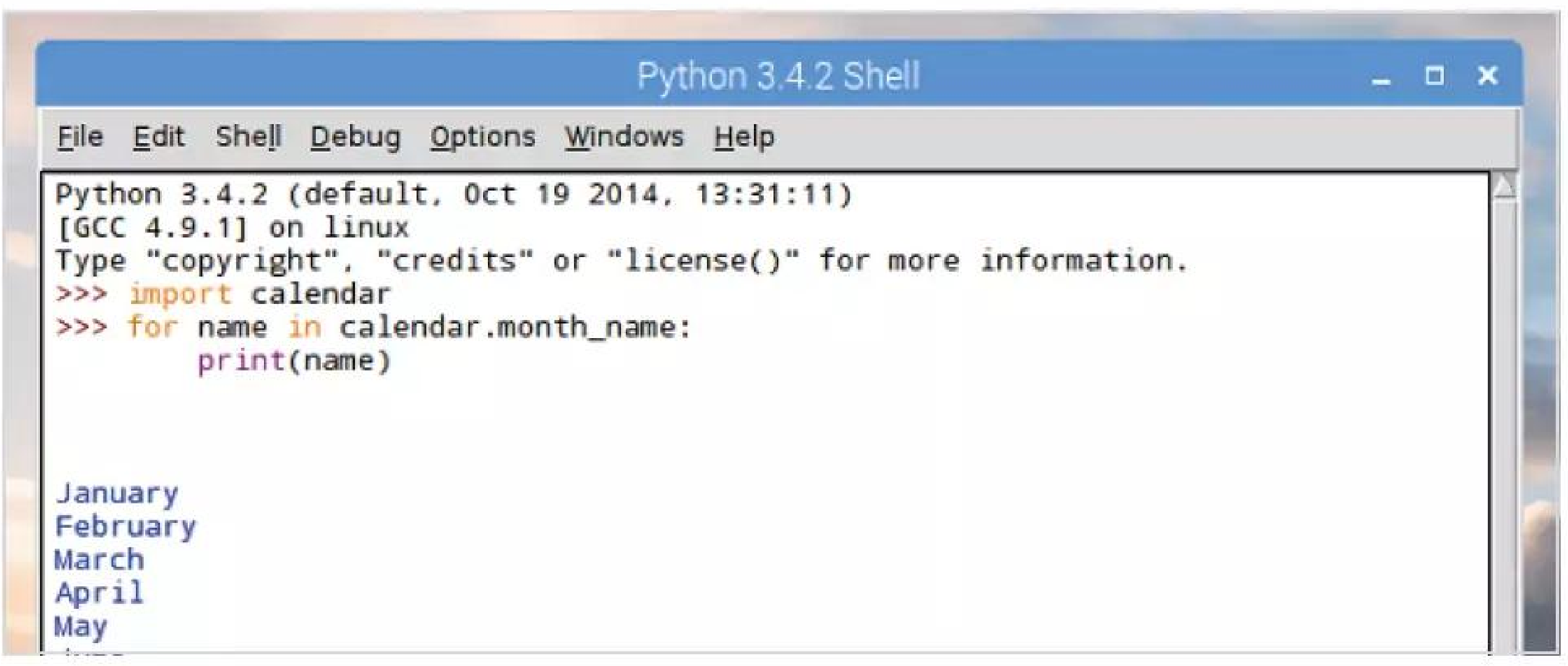
```
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
0
0
0
0
0
1
2
3
4
5
6
7
8
9
10
11
12
```

STEP 8

You're also able to print the individual months or days of the week:

```
import calendar
for name in calendar.month_name:
   print(name)
```

```
import calendar
for name in calendar.day_name:
   print(name)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

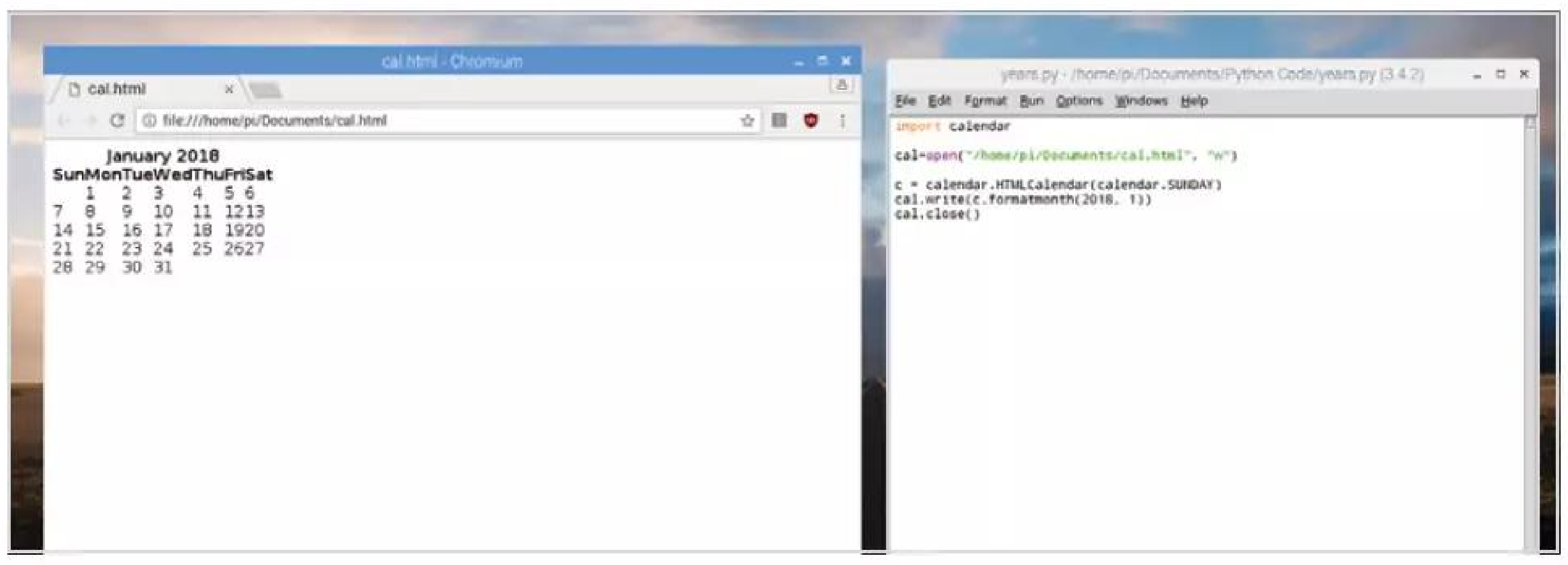
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import calendar
>>> for name in calendar.month_name:
        print(name)



January
February
March
April
May
```

STEP 9

The calendar module also allows us to write the functions in HTML, so that you can display it on a website. Let's start by creating a new file:

```
import calendar
cal=open("/home/pi/Documents/cal.html", "w")
c=calendar.HTMLCalendar(calendar.SUNDAY)
cal.write(c.formatmonth(2018, 1))
cal.close()
```

This code will create an HTML file called cal, open it with a browser and it displays the calendar for January 2018.


cal.html - Chromium
cal.html
file:///home/pi/Documents/cal.html
years.py - /home/pi/Documents/Python Code/years.py (3.4.2)
File Edit Format Run Options Windows Help

```
      January 2018
SunMonTueWedThuFriSat
   1   2   3   4  5 6
7  8   9   10  11 1213
14 15  16  17  18 1920
21 22  23  24  25 2627
28 29  30  31
```

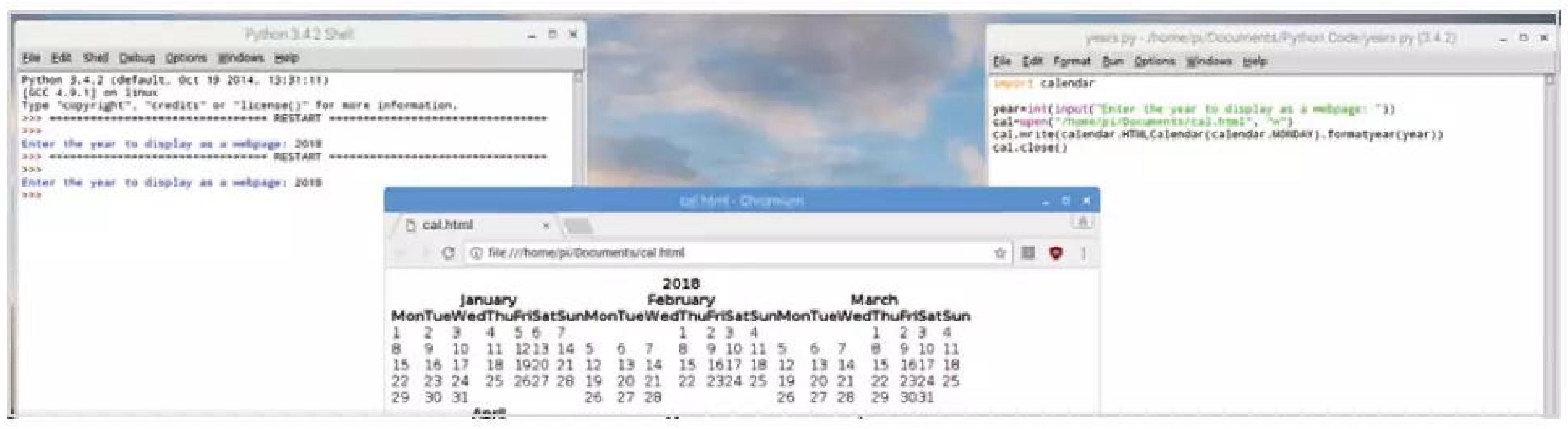
```
import calendar

cal=open("/home/pi/Documents/cal.html", "w")

c = calendar.HTMLCalendar(calendar.SUNDAY)
cal.write(c.formatmonth(2018, 1))
cal.close()
```

STEP 10

Of course, you can modify that to display a given year as a web page calendar:

```
import calendar
year=int(input("Enter the year to display as a webpage: "))
cal=open("/home/pi/Documents/cal.html", "w")
cal.write(calendar.HTMLCalendar(calendar.MONDAY).formatyear(year))
cal.close()
```

This code asks the user for a year, then creates the necessary webpage. Remember to change your file destination.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
years.py - /home/pi/Documents/Python Code/years.py (3.4.2)
File Edit Format Run Options Windows Help
cal.html - Chromium
cal.html
file:///home/pi/Documents/cal.html

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter the year to display as a webpage: 2018
>>> ================================ RESTART ================================
>>>
Enter the year to display as a webpage: 2018
>>>
```

```
import calendar

year=int(input("Enter the year to display as a webpage: "))
cal=open("/home/pi/Documents/cal.html", "w")
cal.write(calendar.HTMLCalendar(calendar.MONDAY).formatyear(year))
cal.close()
```

```
                                  2018
         January                   February                     March
MonTueWedThuFriSatSunMonTueWedThuFriSatSunMonTueWedThuFriSatSun
1   2   3   4  5 6  7               1  2 3  4               1  2 3  4
8   9   10  11 1213 14 5   6   7   8  9 10 11 5   6   7   8  9 10 11
15  16  17  18 1920 21 12  13  14  15 1617 18 12  13  14  15 1617 18
22  23  24  25 2627 28 19  20  21  22 2324 25 19  20  21  22 2324 25
29  30  31             26  27  28             26  27  28  29 3031
```

OS Module

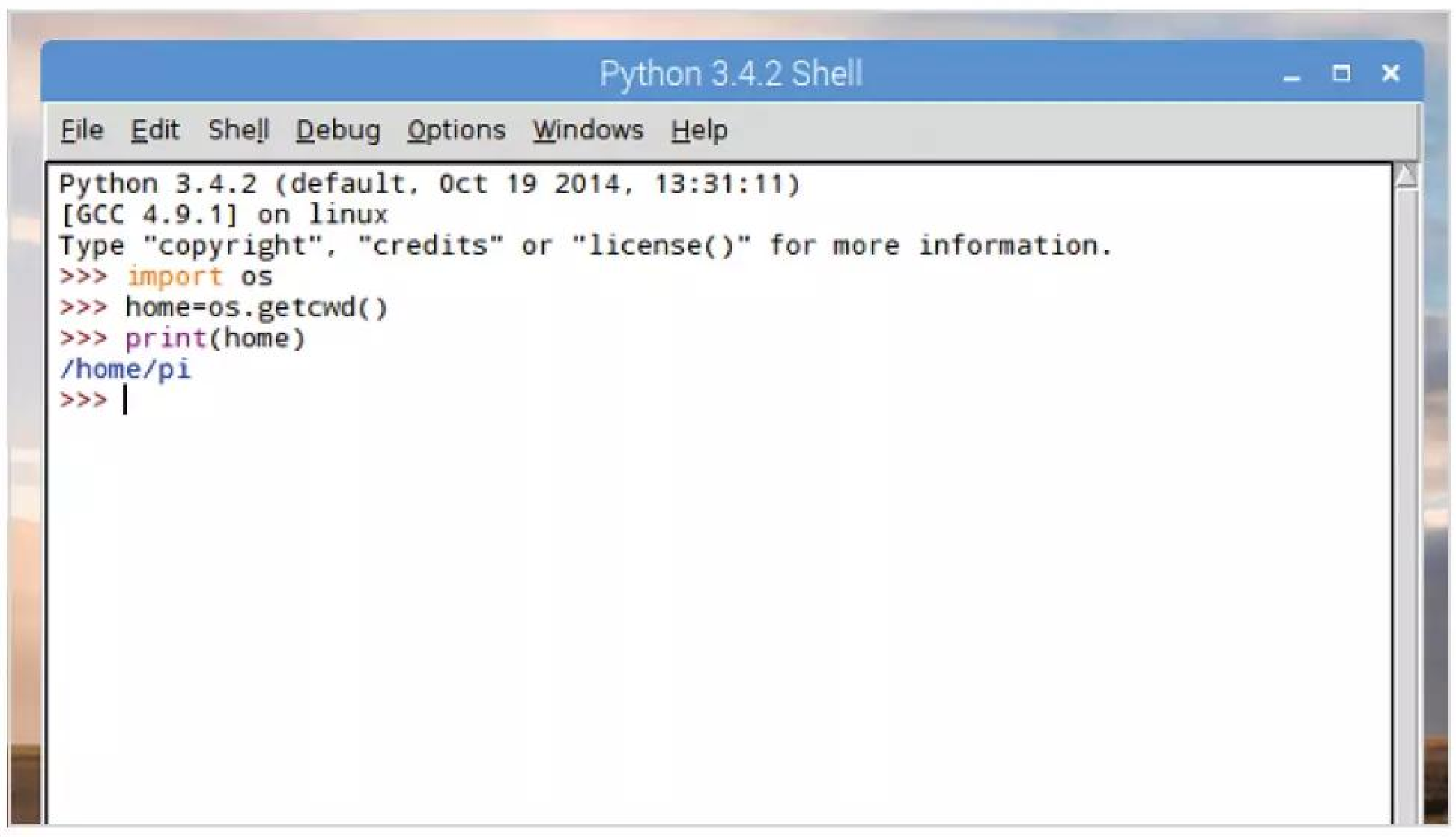
The OS module allows you to interact directly with the built-in commands found in your operating system. Commands vary depending on the OS you're running, as some will work with Windows whereas others will work with Linux and macOS.

INTO THE SYSTEM

One of the primary features of the OS module is the ability to list, move, create, delete and otherwise interact with files stored on the system, making it the perfect module for backup code.

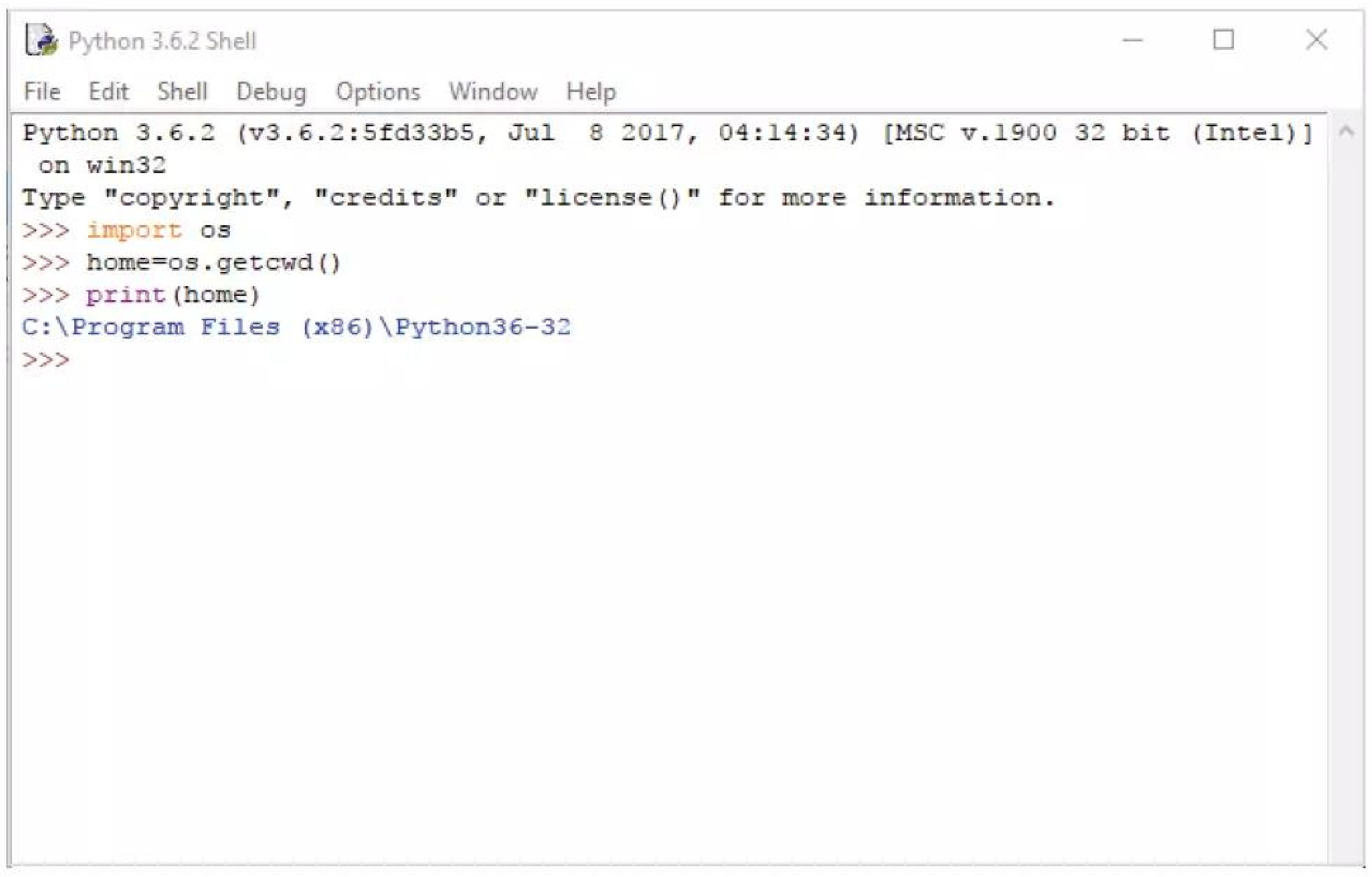
STEP 1 You can start the OS module with some simple functions to see how it interacts with the operating system environment that Python is running on. If you're using Linux or the Raspberry Pi, try this:

```
import os
home=os.getcwd()
print(home)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

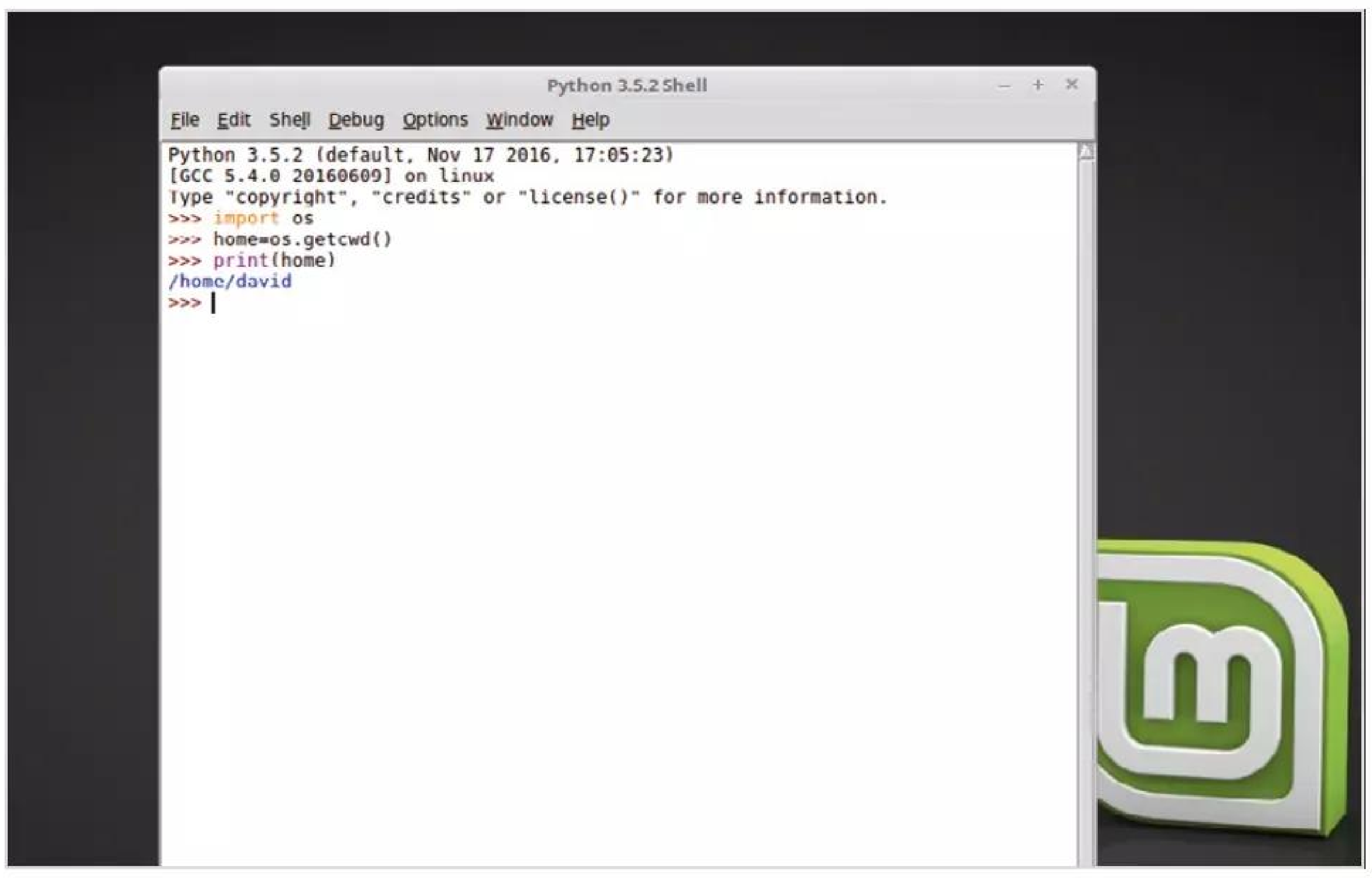
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> home=os.getcwd()
>>> print(home)
/home/pi
>>> |
```

STEP 2 The returned result from printing the variable home is the current user's home folder on the system. In our example that's /home/pi; it will be different depending on the user name you log in as and the operating system you use. For example, Windows 10 will output: C:\Program Files (x86)\Python36-32.


Python 3.6.2 Shell
File Edit Shell Debug Options Window Help

```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> home=os.getcwd()
>>> print(home)
C:\Program Files (x86)\Python36-32
>>>
```

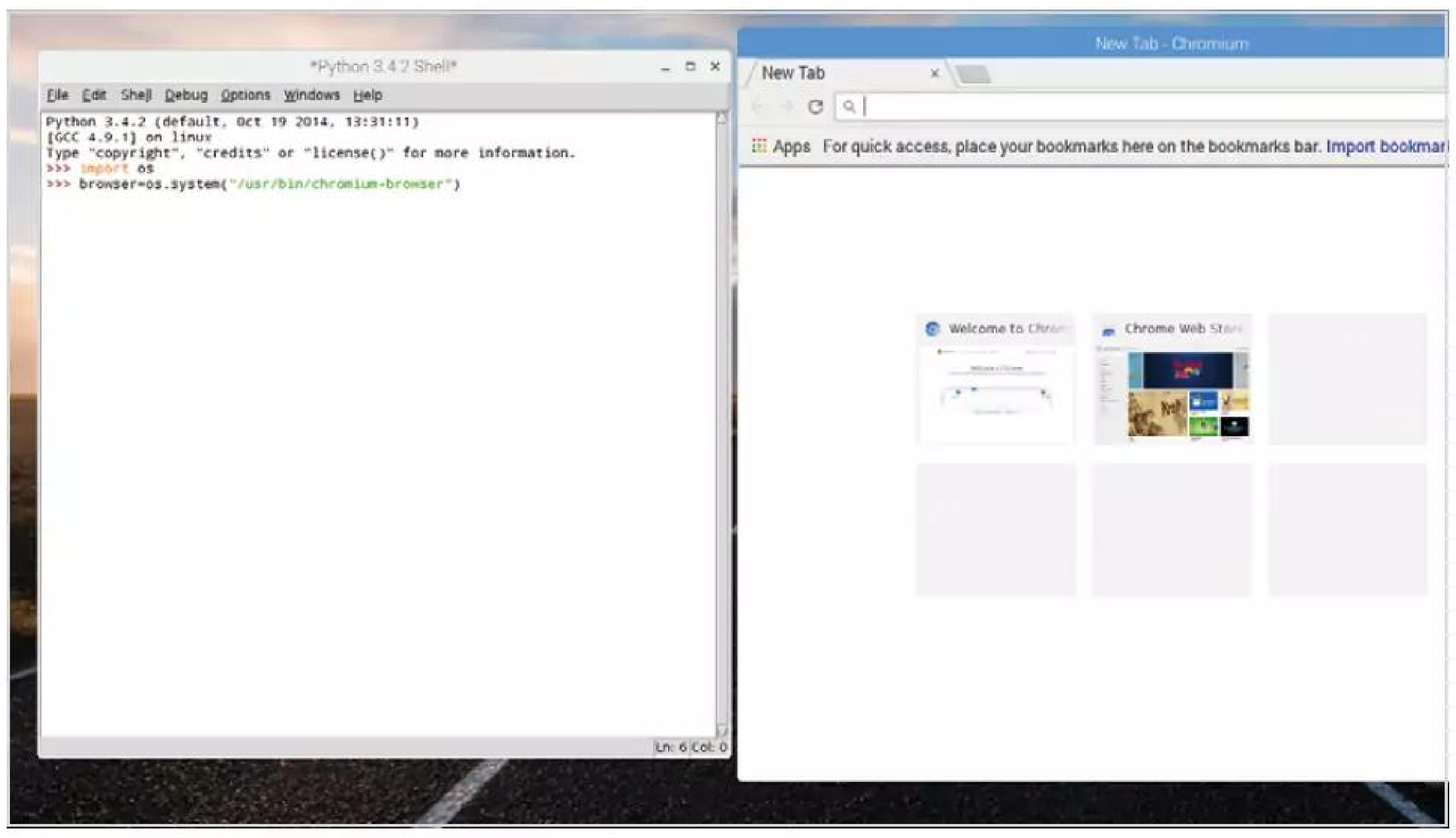
STEP 3 The Windows output is different as that's the current working directory of Python, as determined by the system; as you might suspect, the os.getcwd() function is asking Python to retrieve the Current Working Directory. Linux users will see something along the same lines as the Raspberry Pi, as will macOS users.


Python 3.5.2 Shell
File Edit Shell Debug Options Window Help

```
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> home=os.getcwd()
>>> print(home)
/home/david
>>> |
```

STEP 4 Yet another interesting element to the OS module, is its ability to launch programs that are installed in the host system. For instance, if you wanted to launch the Chromium browser from within a Python program you can use the command:

```
import os
browser=os.system("/usr/bin/chromium-browser")
```

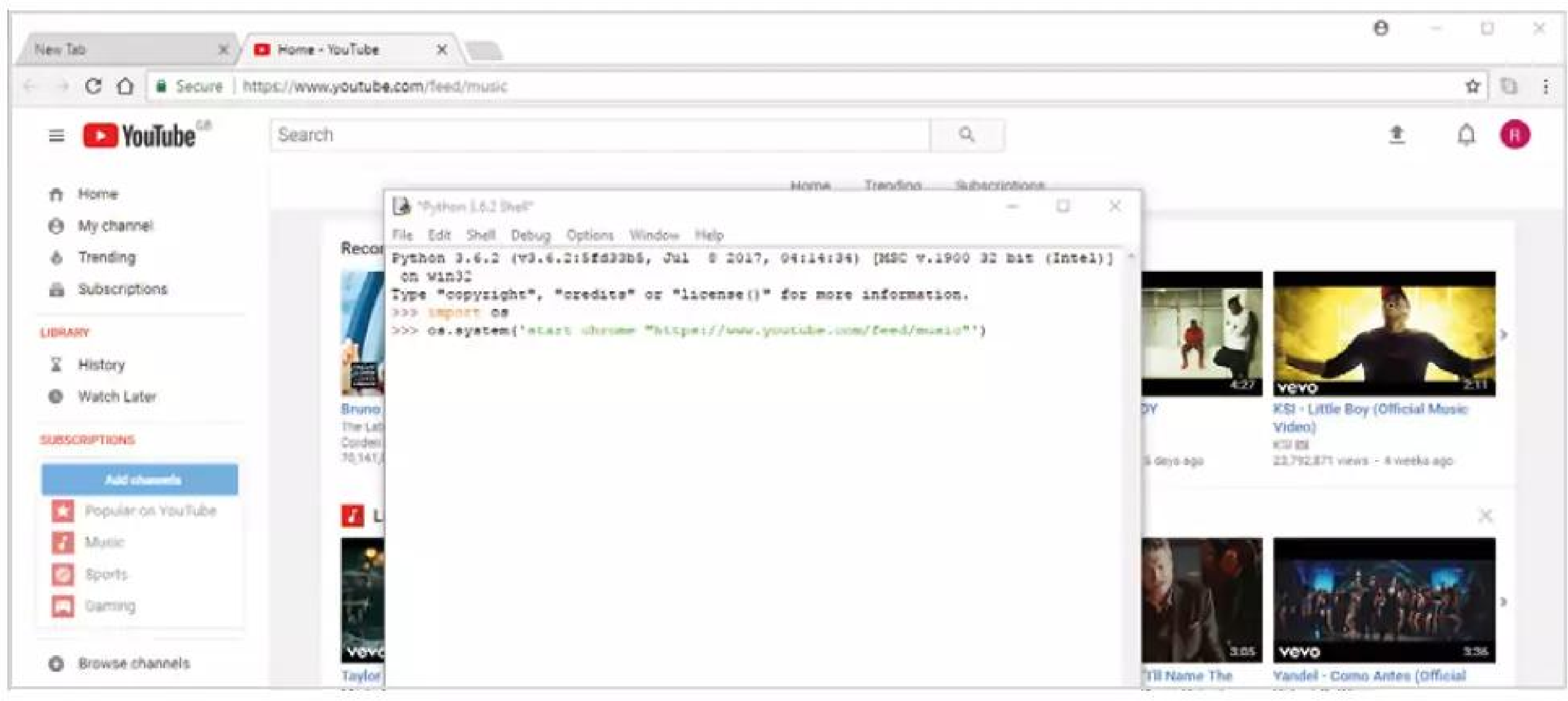

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
New Tab - Chromium
New Tab
Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmar
Welcome to Chrom
Chrome Web Store
Ln: 6 Col: 0

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> browser=os.system("/usr/bin/chromium-browser")
```

STEP 5

The os.system() function is what allows interaction with external programs; you can even call up previous Python programs using this method. You will obviously need to know the full path and program file name for it to work successfully. However, you can use the following:

```
import os
os.system('start chrome "https://www.youtube.com/
feed/music"')
```

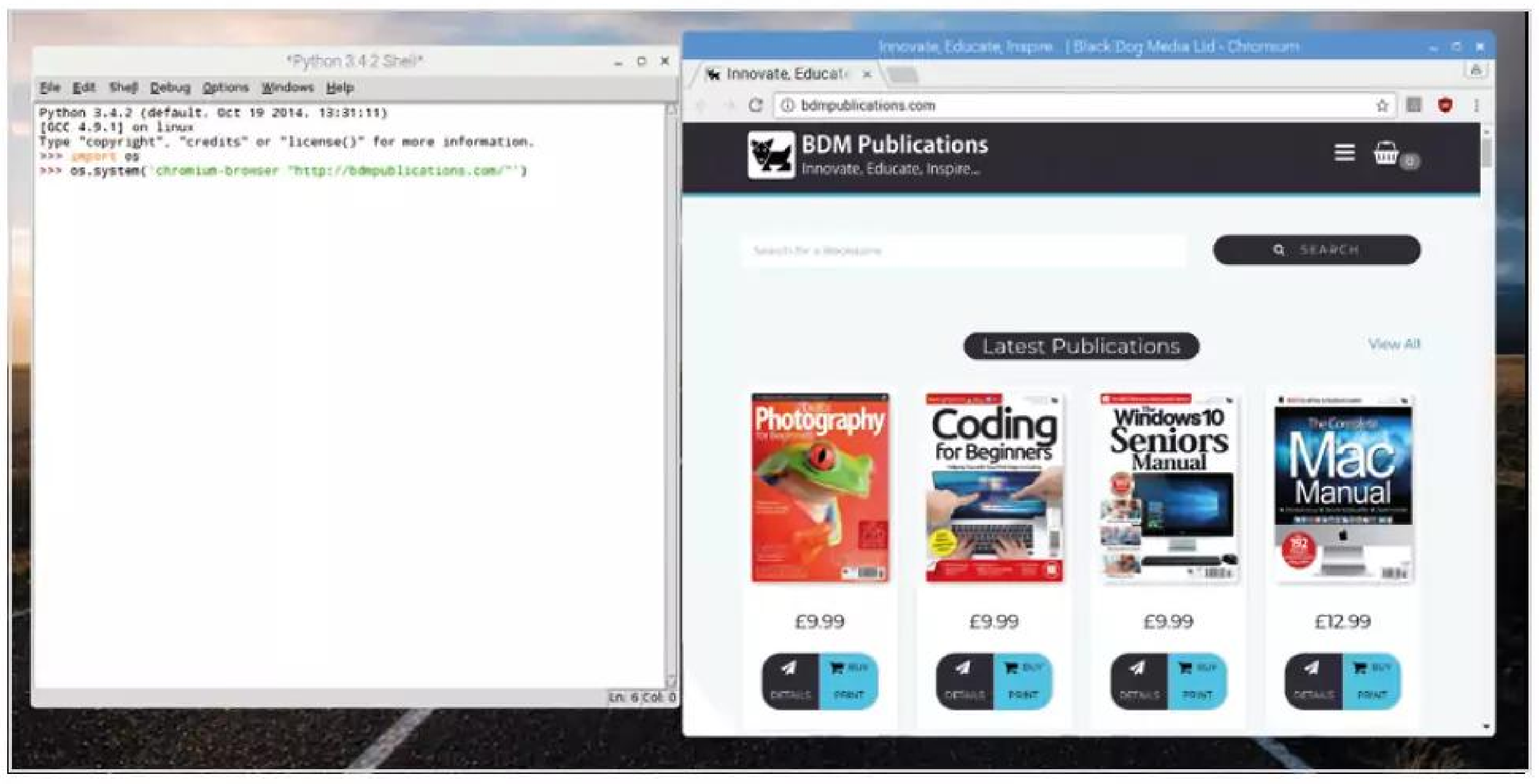


```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.system('start chrome "https://www.youtube.com/feed/music"')
```

STEP 6

For Step 5's example we used Windows, to show that the OS module works roughly the same across all platforms. In that case, we opened YouTube's music feed page, so it is therefore possible to open specific pages:

```
import os
os.system('chromium-browser "http://
bdmpublications.com/"')
```

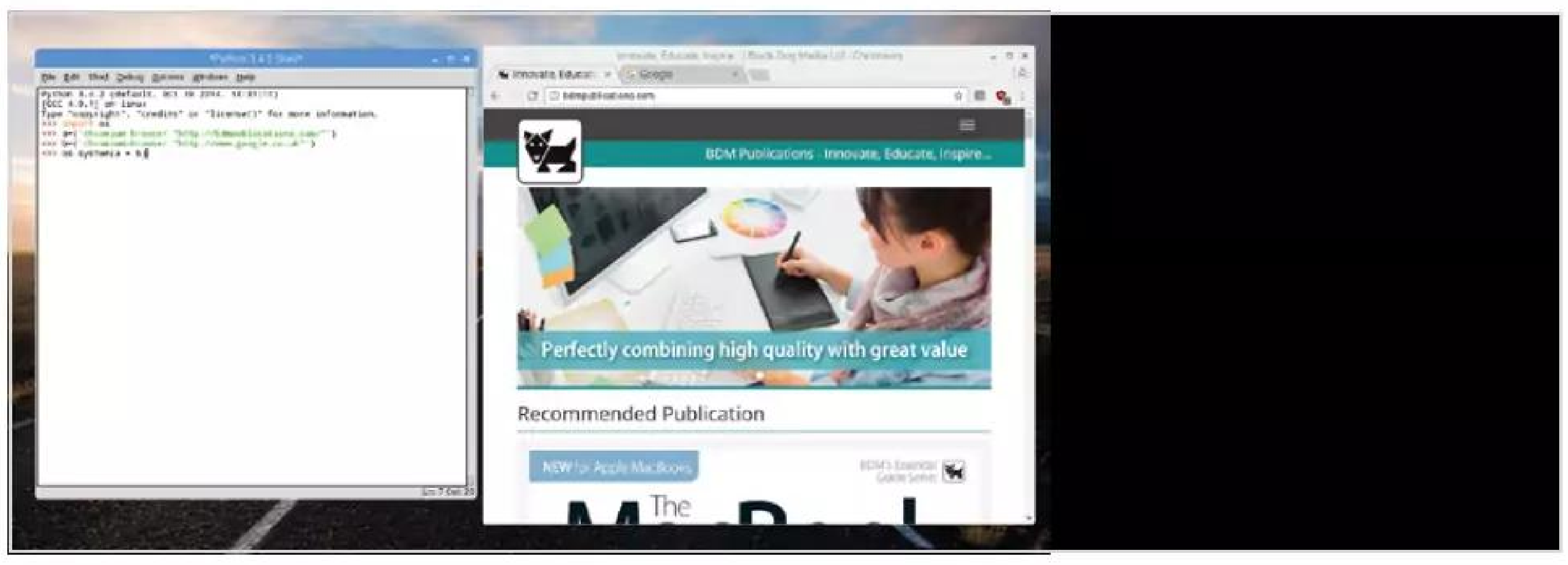


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.system('chromium-browser "http://bdmpublications.com/"')
```

STEP 7

Note in the previous step's example the use of single and double-quotes. The single quotes encase the entire command and launching Chromium, whereas the double quotes open the specified page. You can even use variables to call multiple tabs in the same browser:

```
import os
a=('chromium-browser "http://bdmpublications.
com/"')
b=('chromium-browser "http://www.google.co.uk"')
os.system(a + b)
```

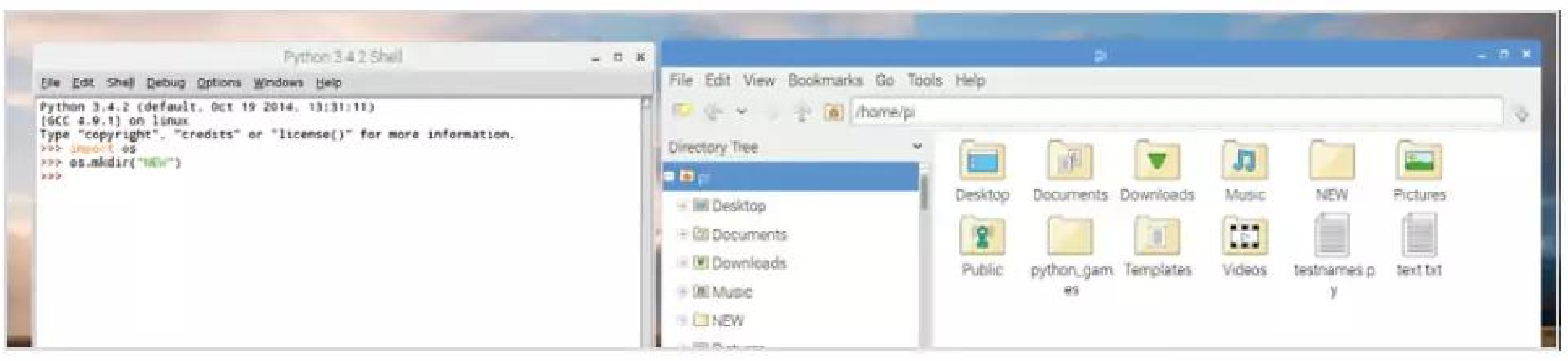


STEP 8

The ability to manipulate directories, or folders if you prefer, is one of the OS module's best features. For example, to create a new directory you can use:

```
import os
os.mkdir("NEW")
```

This creates a new directory within the Current Working Directory, named according to the object in the mkdir function.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.mkdir("NEW")
>>>
```

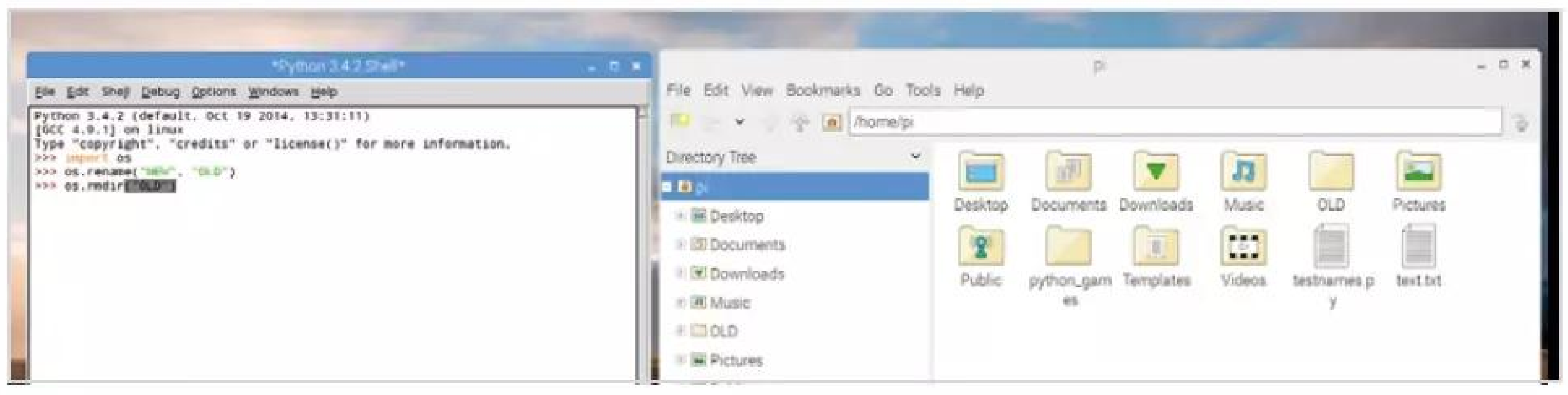
STEP 9

You can also rename any directories you've created by entering:

```
import os
os.rename("NEW", "OLD")
```

To delete them:

```
import os
os.rmdir("OLD")
```

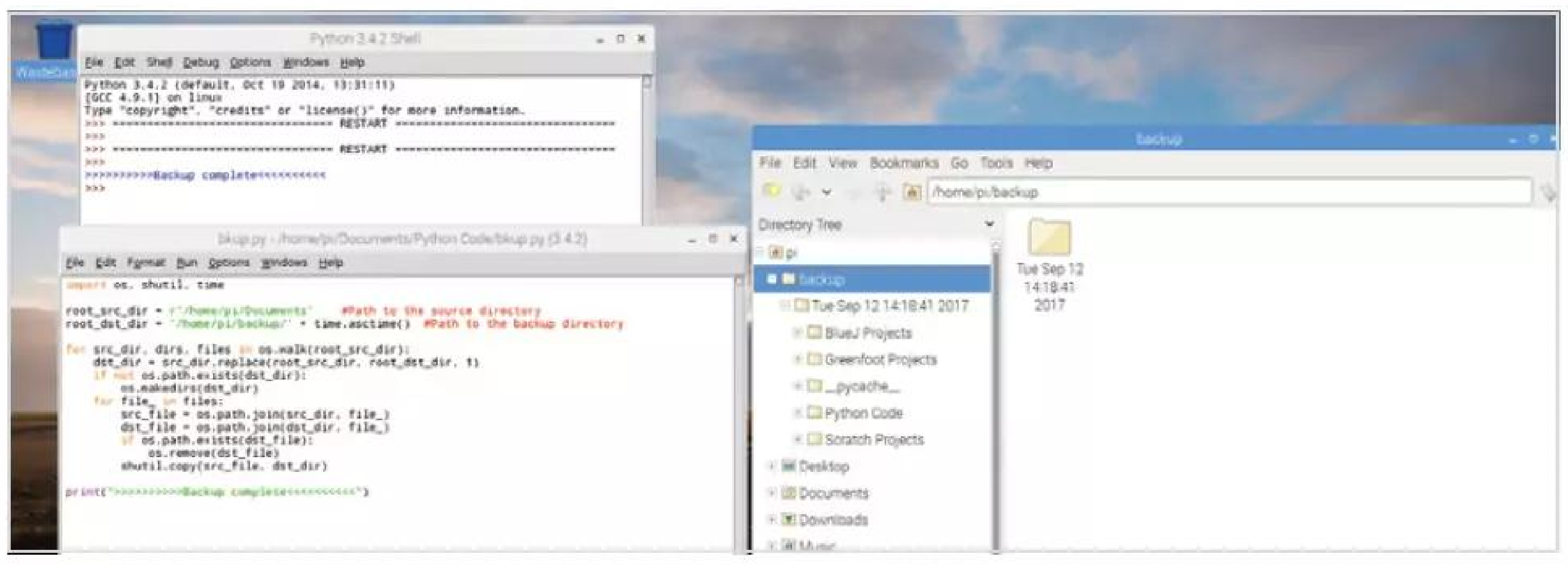


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import os
>>> os.rename("NEW", "OLD")
>>> os.rmdir("OLD")
```

STEP 10

Another module that goes together with OS is shutil. You can use the shutil module together with OS and time to create a time-stamped backup directory, and copy files into it:

```
import os, shutil, time

root_src_dir = r'/home/pi/Documents'
root_dst_dir = '/home/pi/backup/' + time.asctime()

for src_dir, dirs, files in os.walk(root_src_dir):
  dst_dir = src_dir.replace(root_src_dir, root_
dst_dir, 1)
  if not os.path.exists(dst_dir):
    os.makedirs(dst_dir)
  for file_ in files:
    src_file = os.path.join(src_dir, file_)
    dst_file = os.path.join(dst_dir, file_)
    if os.path.exists(dst_file):
      os.remove(dst_file)
    shutil.copy(src_file, dst_dir)

print(">>>>>>>>>>Backup complete<<<<<<<<<<")
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>> ================================ RESTART ================================
>>>
>>>>>>>>>>Backup complete<<<<<<<<<<
>>>
```

```
import os, shutil, time

root_src_dir = r'/home/pi/Documents'    #Path to the source directory
root_dst_dir = '/home/pi/backup/' + time.asctime()  #Path to the backup directory

for src_dir, dirs, files in os.walk(root_src_dir):
    dst_dir = src_dir.replace(root_src_dir, root_dst_dir, 1)
    if not os.path.exists(dst_dir):
        os.makedirs(dst_dir)
    for file_ in files:
        src_file = os.path.join(src_dir, file_)
        dst_file = os.path.join(dst_dir, file_)
        if os.path.exists(dst_file):
            os.remove(dst_file)
        shutil.copy(src_file, dst_dir)

print(">>>>>>>>>>Backup complete<<<<<<<<<<")
```

Random Module

The random module is one you will likely come across many times in your Python programming lifetime; as the name suggests, it's designed to create random numbers or letters. However, it's not exactly random but it will suffice for most needs.

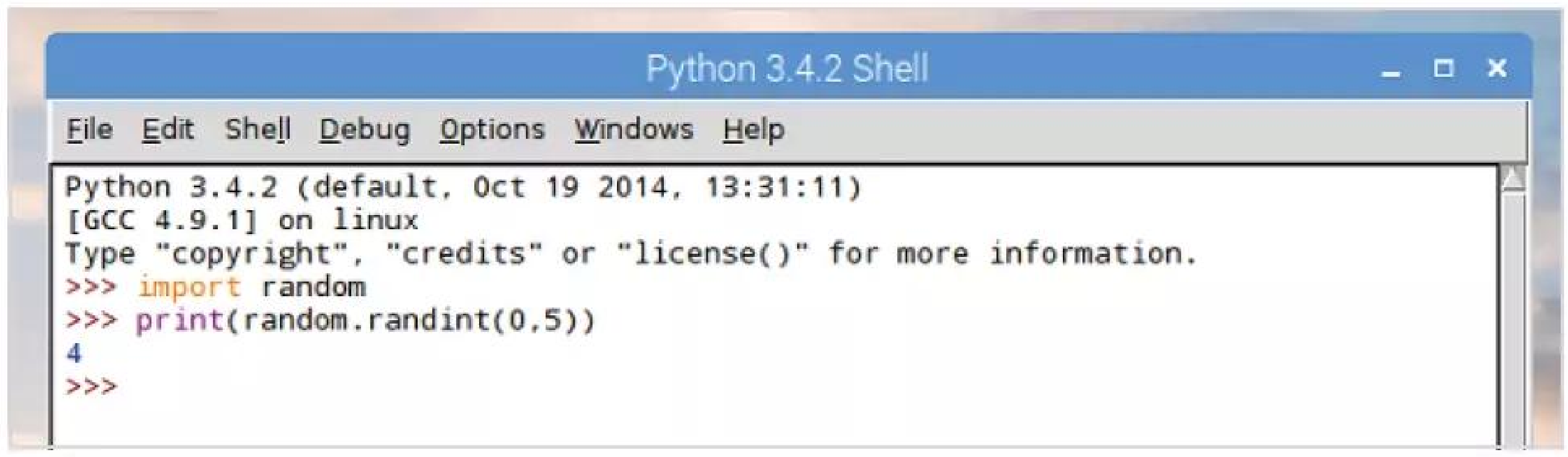
RANDOM NUMBERS

There are numerous functions within the random module, which when applied can create some interesting and very useful Python programs.

STEP 1

Just as with other modules you need to import random before you can use any of the functions we're going to look at in this tutorial. Let's begin by simply printing a random number from 1 to 5:

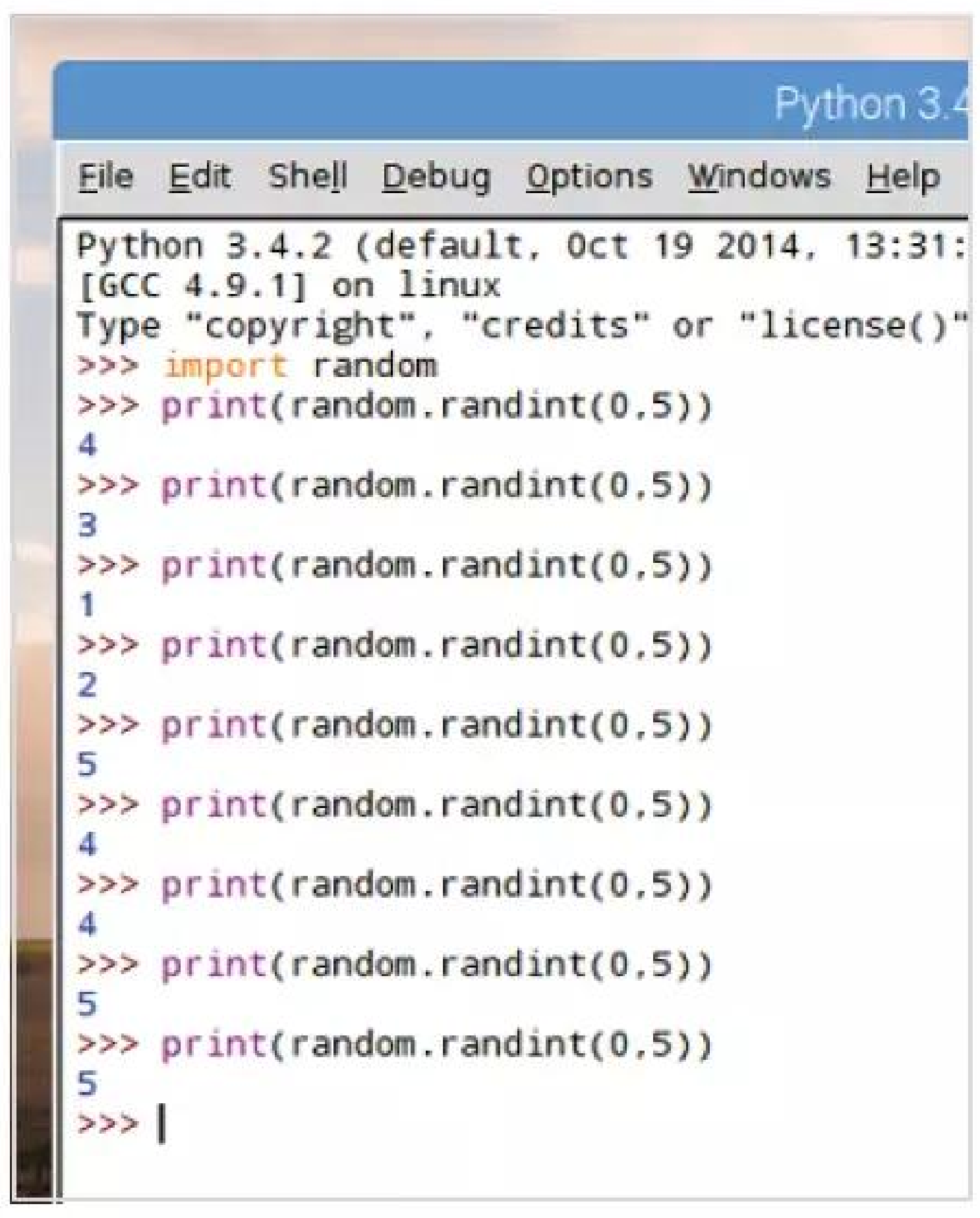
```
import random
print(randomint(0,5))
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> print(random.randint(0,5))
4
>>>
```

STEP 2

In our example the number four was returned. However, enter the print function a few more times and it will display different integer values from the set of numbers given, zero to five. The overall effect, although pseudo-random, is adequate for the average programmer to utilise in their code.



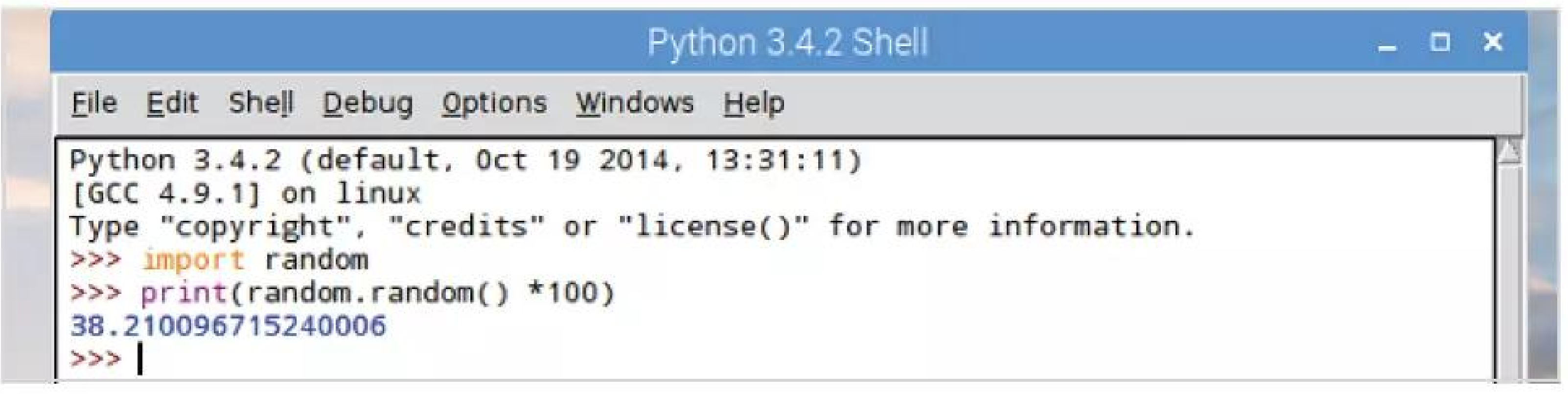
```
Python 3.4.2 (default, Oct 19 2014, 13:31:
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()"
>>> import random
>>> print(random.randint(0,5))
4
>>> print(random.randint(0,5))
3
>>> print(random.randint(0,5))
1
>>> print(random.randint(0,5))
2
>>> print(random.randint(0,5))
5
>>> print(random.randint(0,5))
4
>>> print(random.randint(0,5))
4
>>> print(random.randint(0,5))
5
>>> print(random.randint(0,5))
5
>>> |
```

STEP 3

For a bigger set of numbers, including floating point values, you can extend the range by using the multiplication sign:

```
import random
print(random.random() *100)
```

Will display a floating point number between 0 and 100, to the tune of around fifteen decimal points.



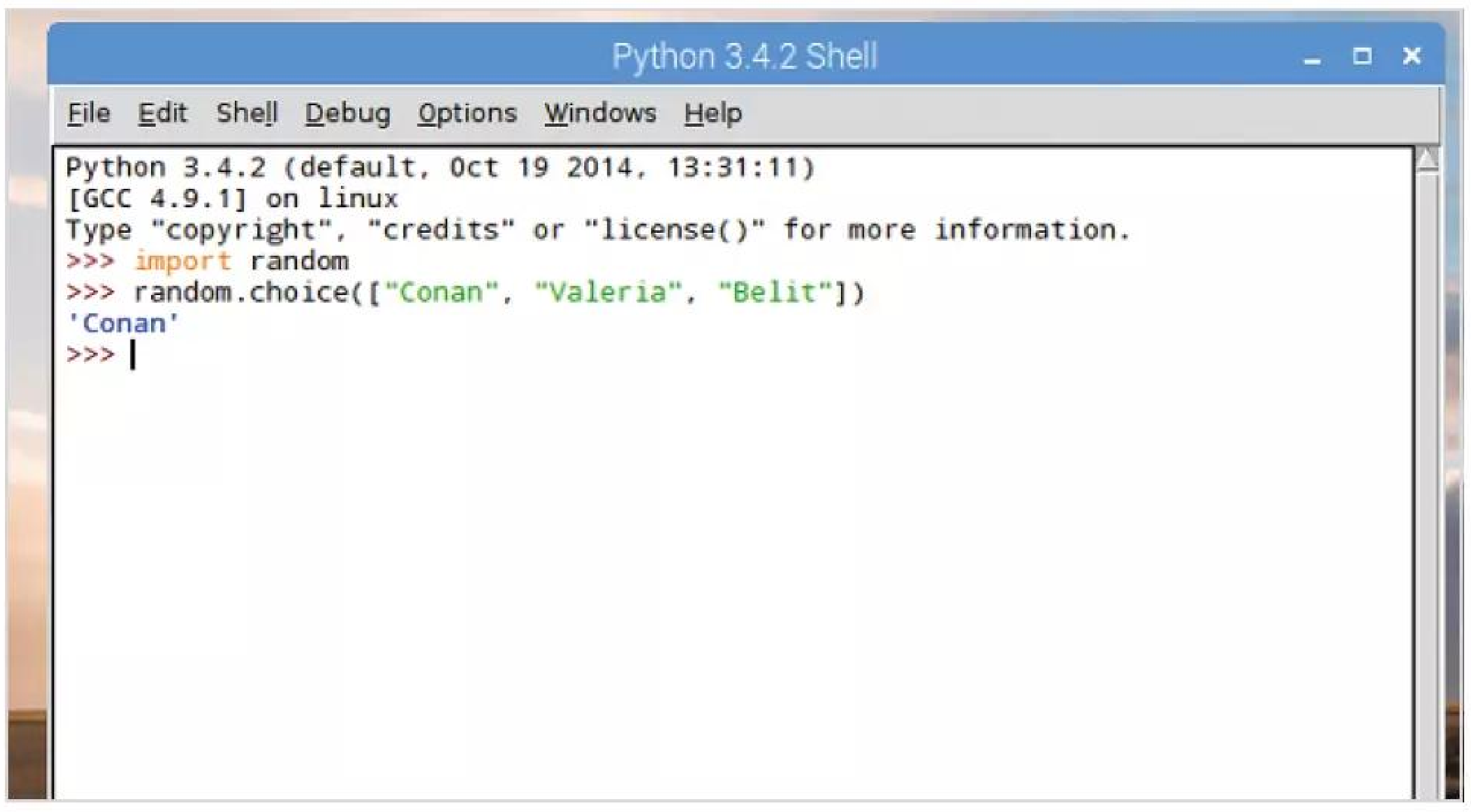
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> print(random.random() *100)
38.210096715240006
>>> |
```

STEP 4

However, the random module isn't used exclusively for numbers. You can use it to select an entry from a list from random, and the list can contain anything:

```
import random
random.choice([“Conan”, “Valeria”, “Belit”])
```

This will display one of the names of our adventurers at random, which is a great addition to a text adventure game.

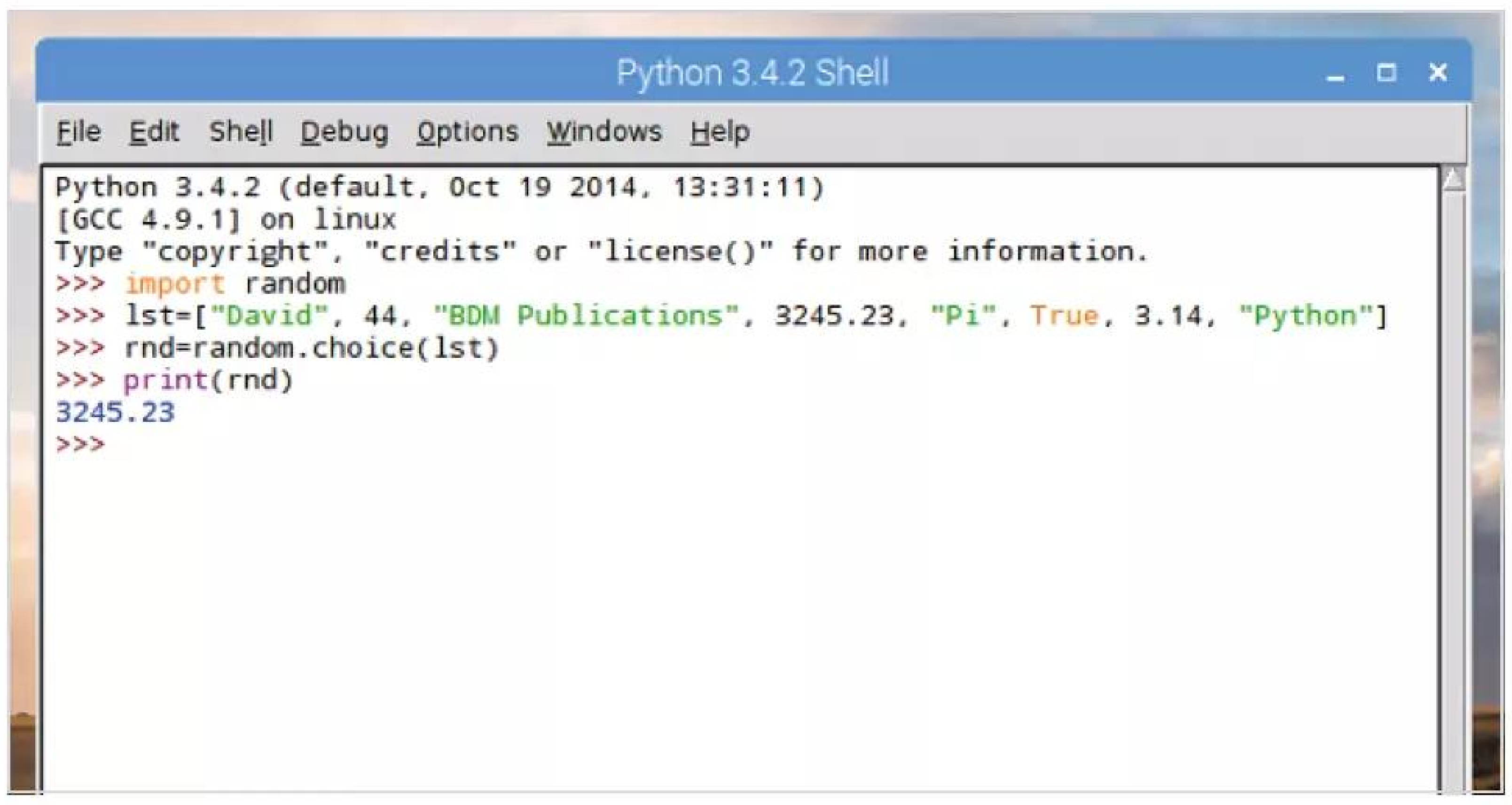


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> random.choice(["Conan", "Valeria", "Belit"])
'Conan'
>>> |
```

STEP 5

You can extend the previous example somewhat by having random.choice() select from a list of mixed variables. For instance:

```
import random
lst=[“David”, 44, “BDM Publications”, 3245.23,
“Pi”, True, 3.14, “Python”]
rnd=random.choice(lst)
print(rnd)
```



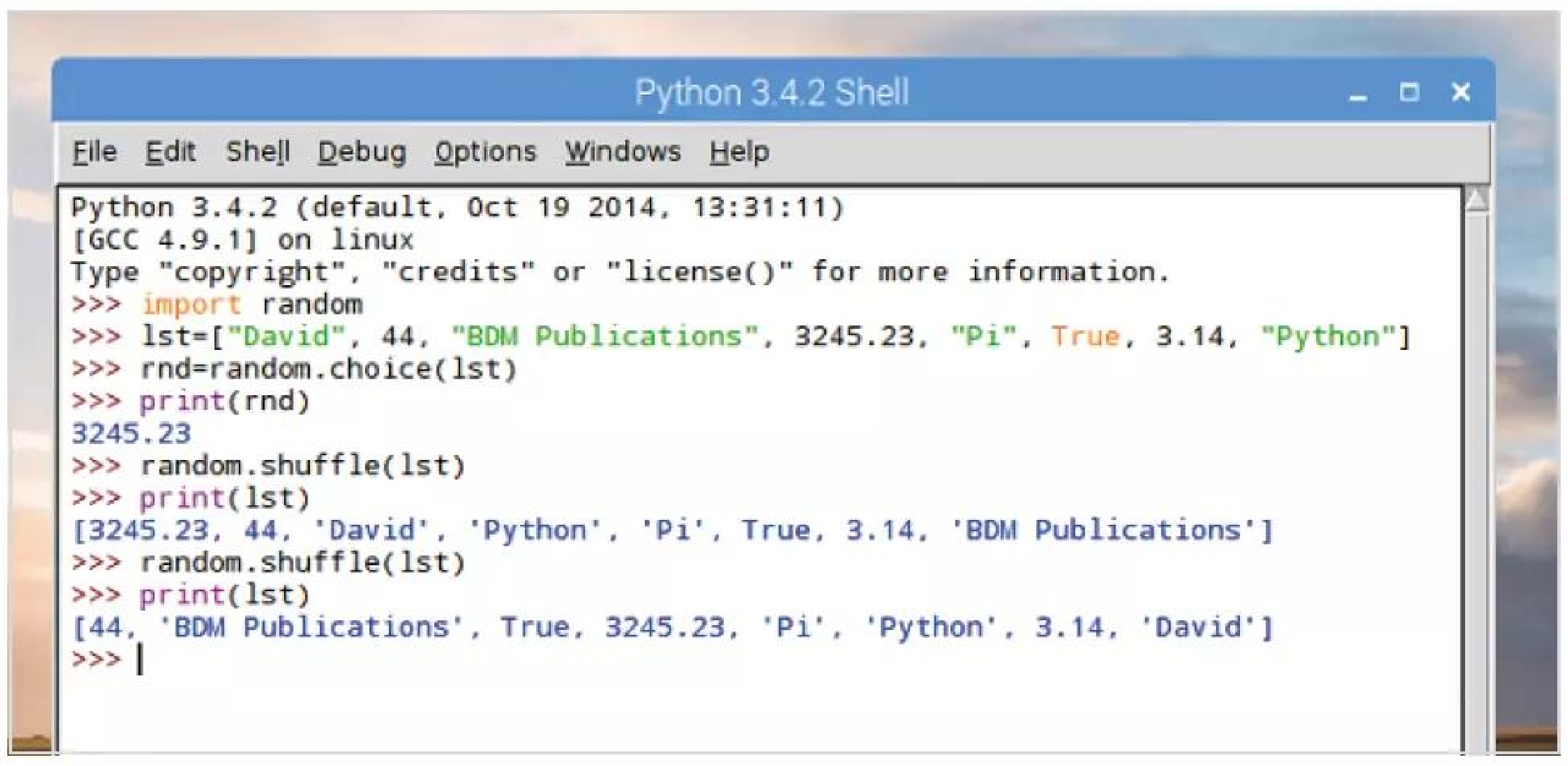
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst=["David", 44, "BDM Publications", 3245.23, "Pi", True, 3.14, "Python"]
>>> rnd=random.choice(lst)
>>> print(rnd)
3245.23
>>>
```

STEP 6

Interestingly, you can also use a function within the random module to shuffle the items in the list, thus adding a little more randomness into the equation:

```
random.shuffle(lst)
print(lst)
```

This way, you can keep shuffling the list before displaying a random item from it.



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

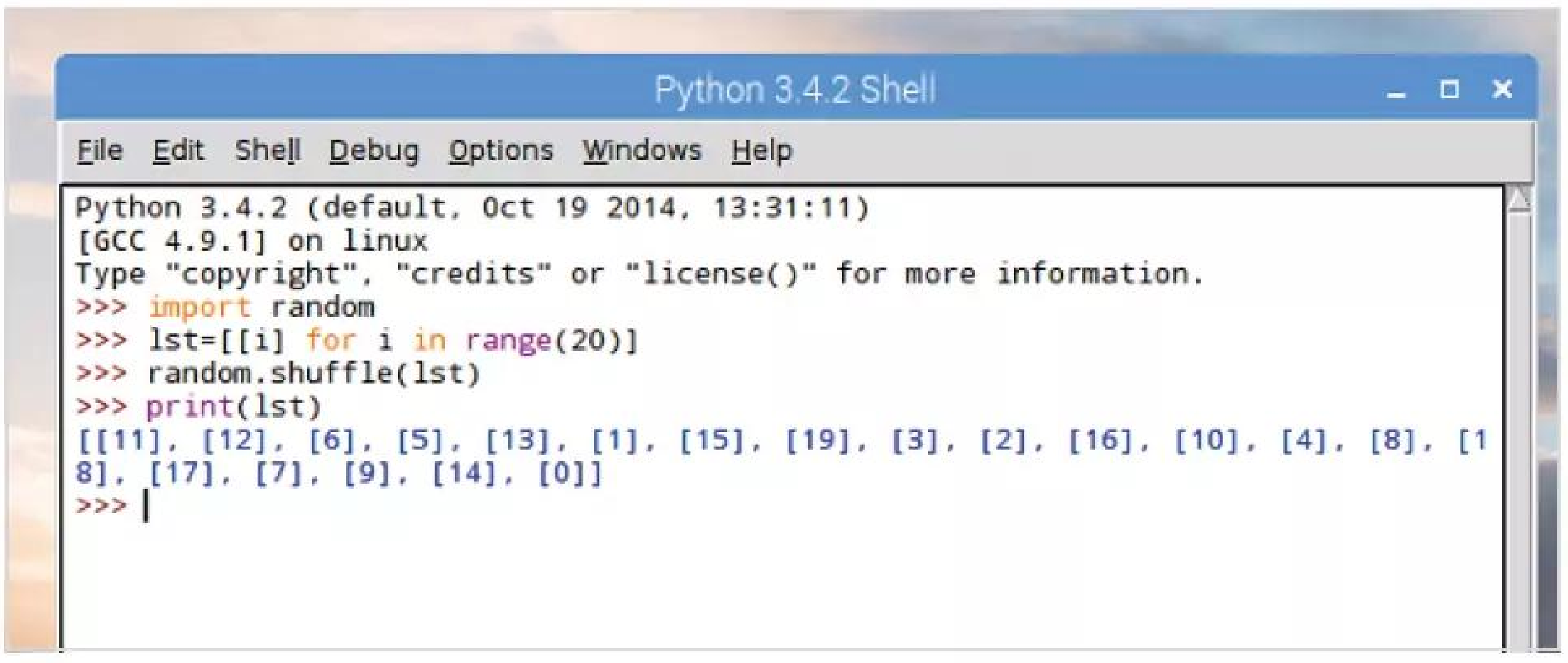
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst=["David", 44, "BDM Publications", 3245.23, "Pi", True, 3.14, "Python"]
>>> rnd=random.choice(lst)
>>> print(rnd)
3245.23
>>> random.shuffle(lst)
>>> print(lst)
[3245.23, 44, 'David', 'Python', 'Pi', True, 3.14, 'BDM Publications']
>>> random.shuffle(lst)
>>> print(lst)
[44, 'BDM Publications', True, 3245.23, 'Pi', 'Python', 3.14, 'David']
>>> |
```

STEP 7

Using shuffle, you can create an entirely random list of numbers. For example, within a given range:

```
import random
lst=[[i] for I in range(20)]
random.shuffle(lst)
print(lst)
```

Keep shuffling the list and you can have a different selection of items from 0 to 20 every time.



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

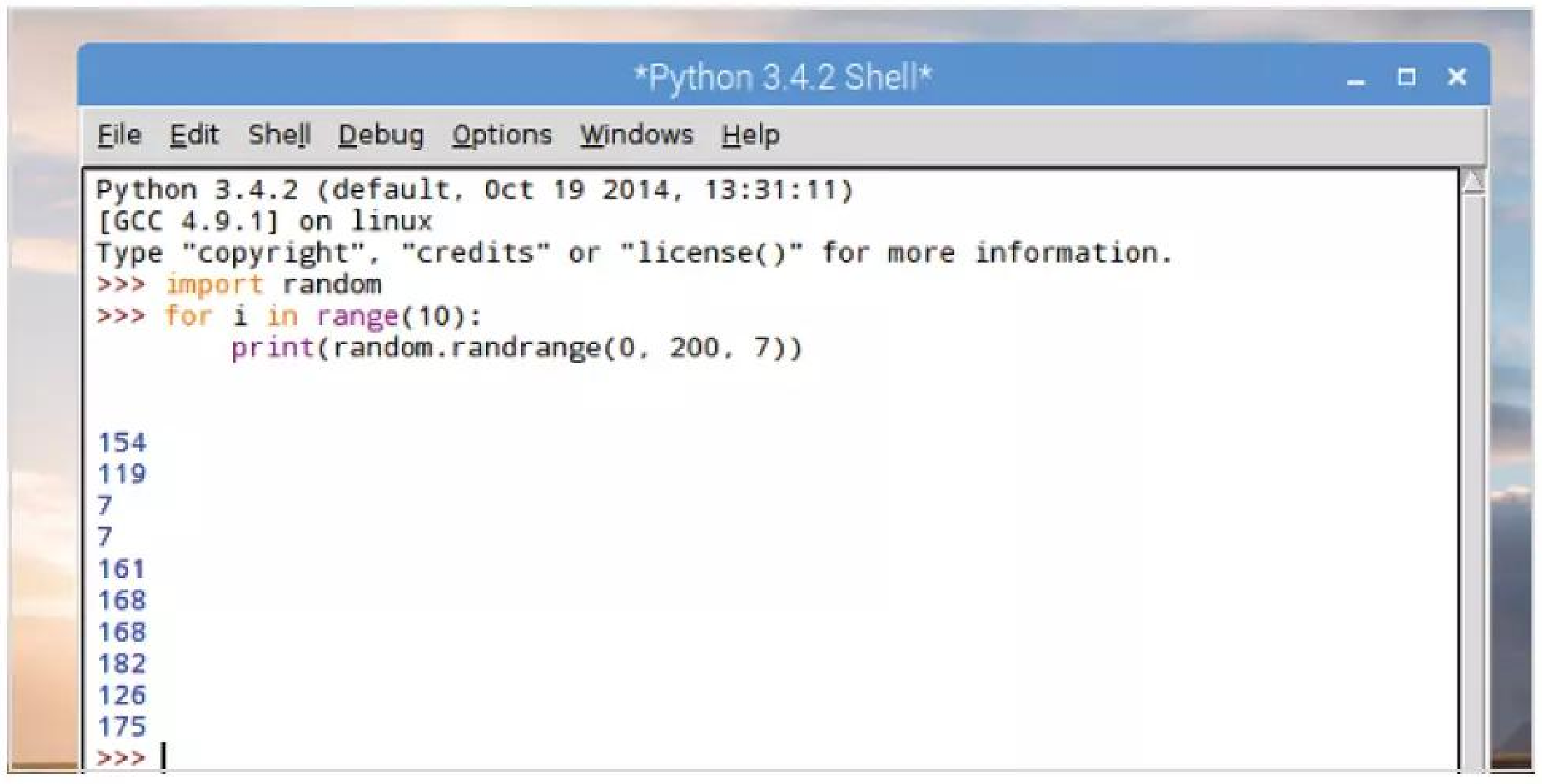
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst=[[i] for i in range(20)]
>>> random.shuffle(lst)
>>> print(lst)
[[11], [12], [6], [5], [13], [1], [15], [19], [3], [2], [16], [10], [4], [8], [1
8], [17], [7], [9], [14], [0]]
>>> |
```

STEP 8

You can also select a random number from a given range in steps, using the start, stop, step loop:

```
import random
for i in range(10):
   print(random.randrange(0, 200, 7))
```

Results will vary but you get the general idea as to how it works.



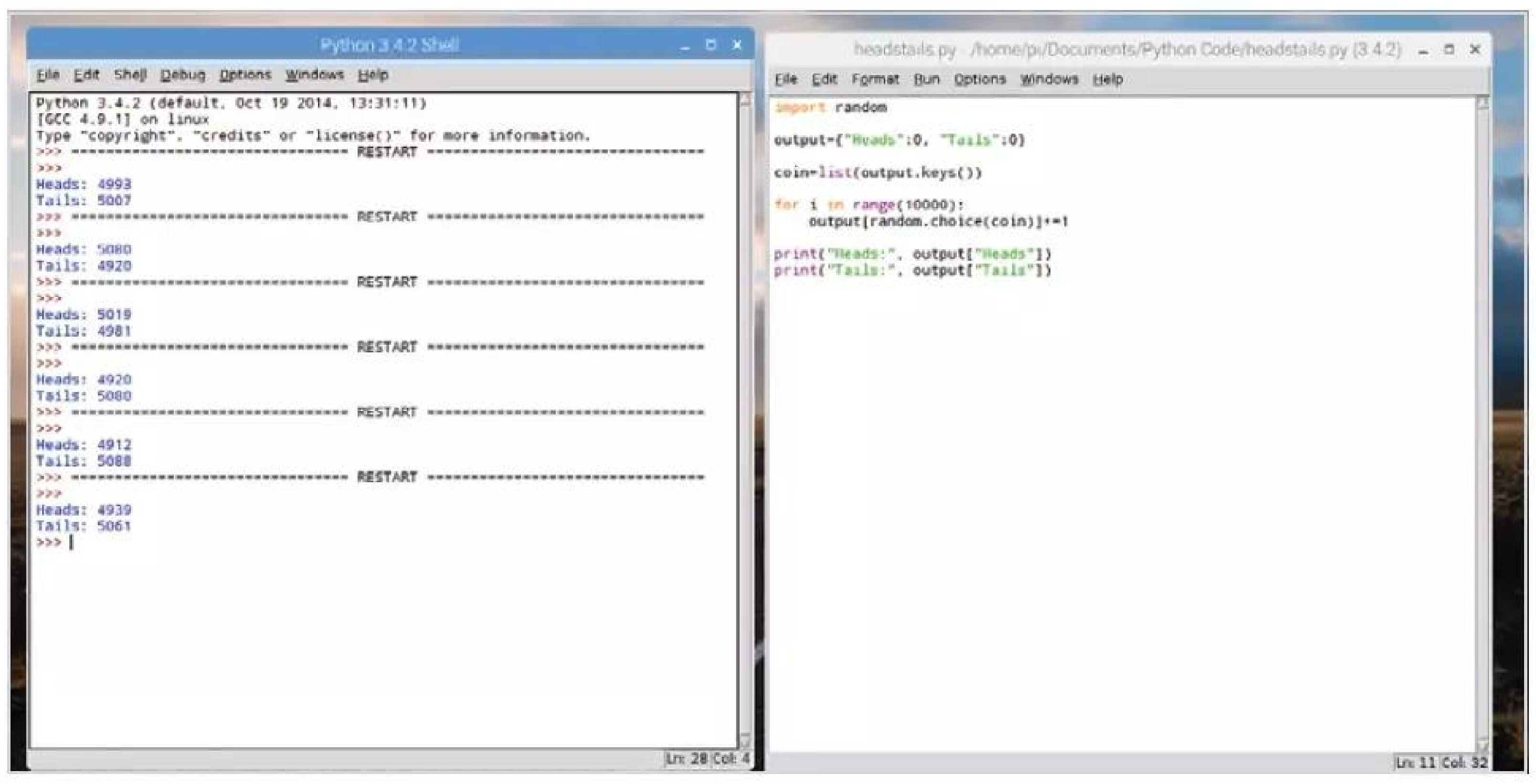
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> for i in range(10):
	print(random.randrange(0, 200, 7))


154
119
7
7
161
168
168
182
126
175
>>> |
```

STEP 9

Let's use an example piece of code which flips a virtual coin ten thousand times and counts how many times it will land on heads or tails:

```
import random
output={"Heads":0, "Tails":0}
coin=list(output.keys())

for i in range(10000):
  output[random.choice(coin)]+=1

print("Heads:", output["Heads"])
print("Tails:", output["Tails"])
```



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 28 Col: 4
headstails.py - /home/pi/Documents/Python Code/headstails.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 11 Col: 32

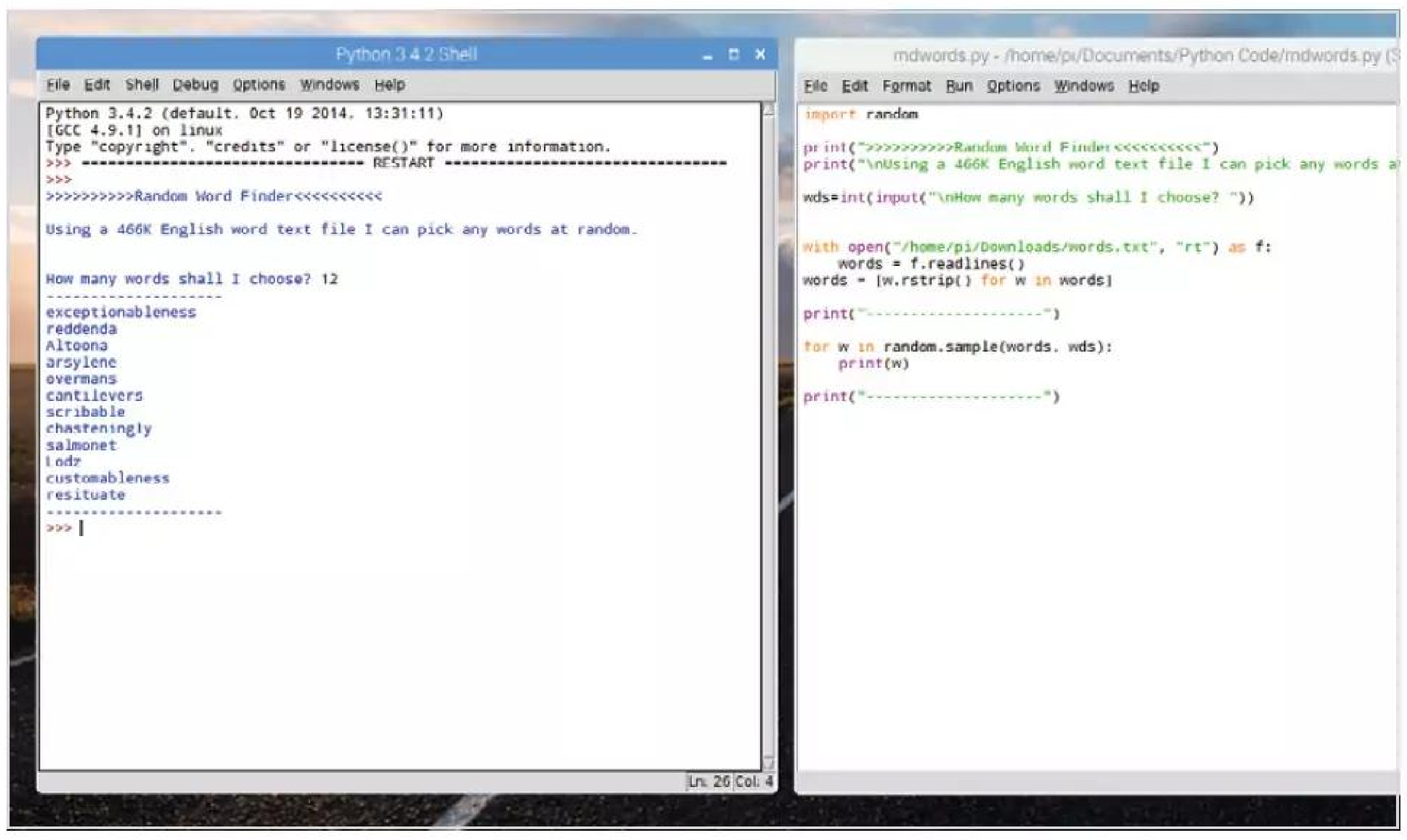
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Heads: 4993
Tails: 5007
>>> ================================ RESTART ================================
>>>
Heads: 5080
Tails: 4920
>>> ================================ RESTART ================================
>>>
Heads: 5019
Tails: 4981
>>> ================================ RESTART ================================
>>>
Heads: 4920
Tails: 5080
>>> ================================ RESTART ================================
>>>
Heads: 4912
Tails: 5088
>>> ================================ RESTART ================================
>>>
Heads: 4939
Tails: 5061
>>> |
```

```
import random

output={"Heads":0, "Tails":0}

coin=list(output.keys())

for i in range(10000):
    output[random.choice(coin)]+=1

print("Heads:", output["Heads"])
print("Tails:", output["Tails"])
```

STEP 10

Here's an interesting piece of code. Using a text file containing 466 thousand words, you can pluck a user generated number of words from the file (text file found at: **www.github.com/dwyl/english-words**):

```
import random

print(">>>>>>>>>>Random Word Finder<<<<<<<<<<")
print("\nUsing a 466K English word text file I can 
pick any words at random.\n")

wds=int(input("\nHow many words shall I choose? 
"))

with open("/home/pi/Downloads/words.txt", "rt") as 
f:
  words = f.readlines()
words = [w.rstrip() for w in words]

print("--------------------")

for w in random.sample(words, wds):
  print(w)

print("--------------------")
```



Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 26 Col: 4
rndwords.py - /home/pi/Documents/Python Code/rndwords.py (3
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>>>>>>>>>Random Word Finder<<<<<<<<<<

Using a 466K English word text file I can pick any words at random.


How many words shall I choose? 12
--------------------
exceptionableness
reddenda
Altoona
arsylene
overmans
cantilevers
scribable
chasteningly
salmonet
lodr
customableness
resituate
--------------------
>>> |
```

```
import random

print(">>>>>>>>>>Random Word Finder<<<<<<<<<<")
print("\nUsing a 466K English word text file I can pick any words a

wds=int(input("\nHow many words shall I choose? "))


with open("/home/pi/Downloads/words.txt", "rt") as f:
    words = f.readlines()
words = [w.rstrip() for w in words]

print("--------------------")

for w in random.sample(words, wds):
    print(w)

print("--------------------")
```

Tkinter Module

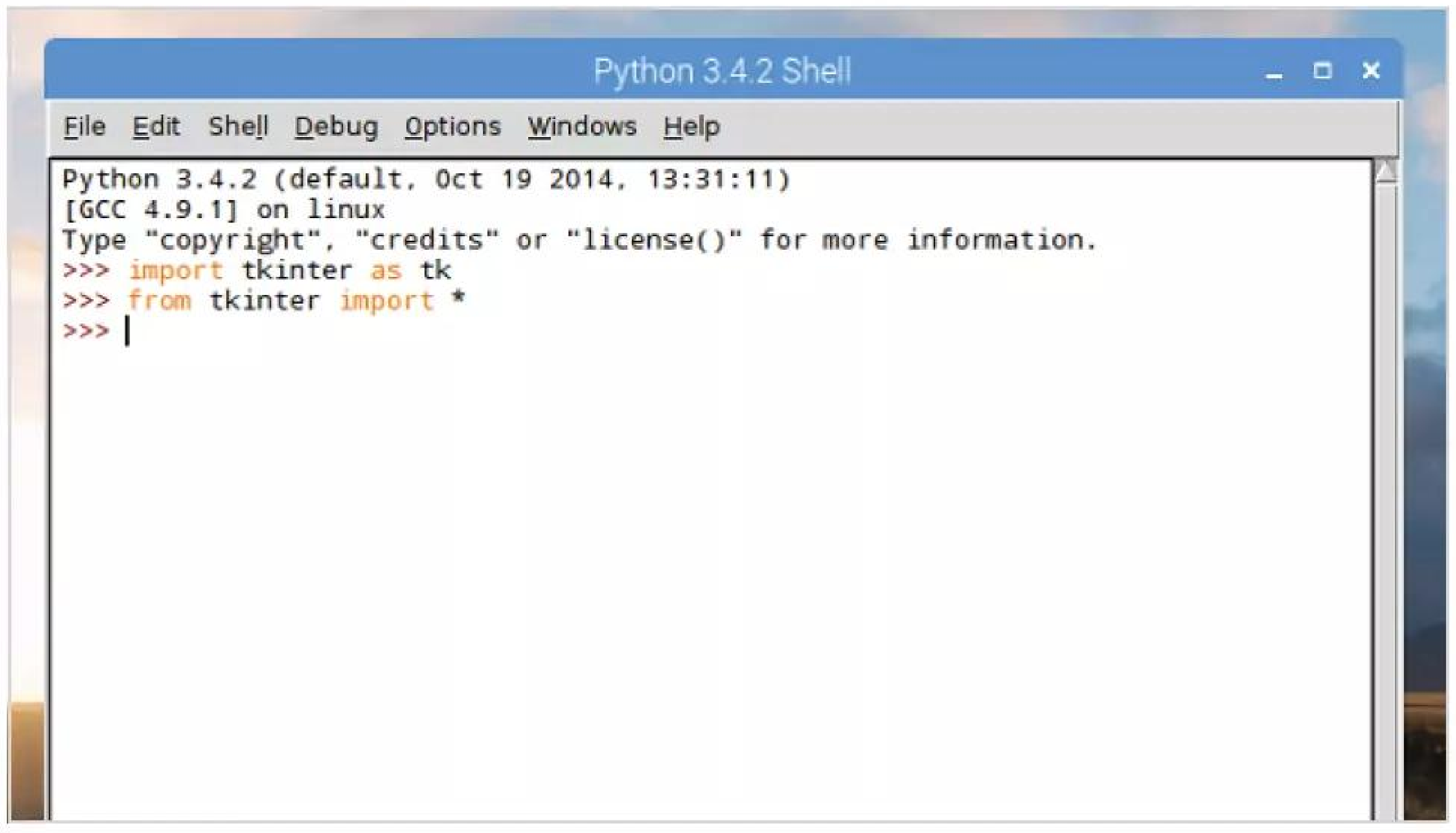
While running your code from the command line, or even in the Shell, is perfectly fine, Python is capable of so much more. The Tkinter module enables the programmer to set up a Graphical User Interface to interact with the user, and it's surprisingly powerful too.

GETTING GUI

Tkinter is easy to use but there's a lot more you can do with it. Let's start by seeing how it works and getting some code into it. Before long you will discover just how powerful this module really is.

STEP 1 Tkinter is usually built into Python 3. However, if it's available when you enter: `import tkinter`, then you need to `pip install tkinter` from the command prompt. We can start to import modules differently than before, to save on typing and by importing all their contents:

```
import tkinter as tk
from tkinter import *
```

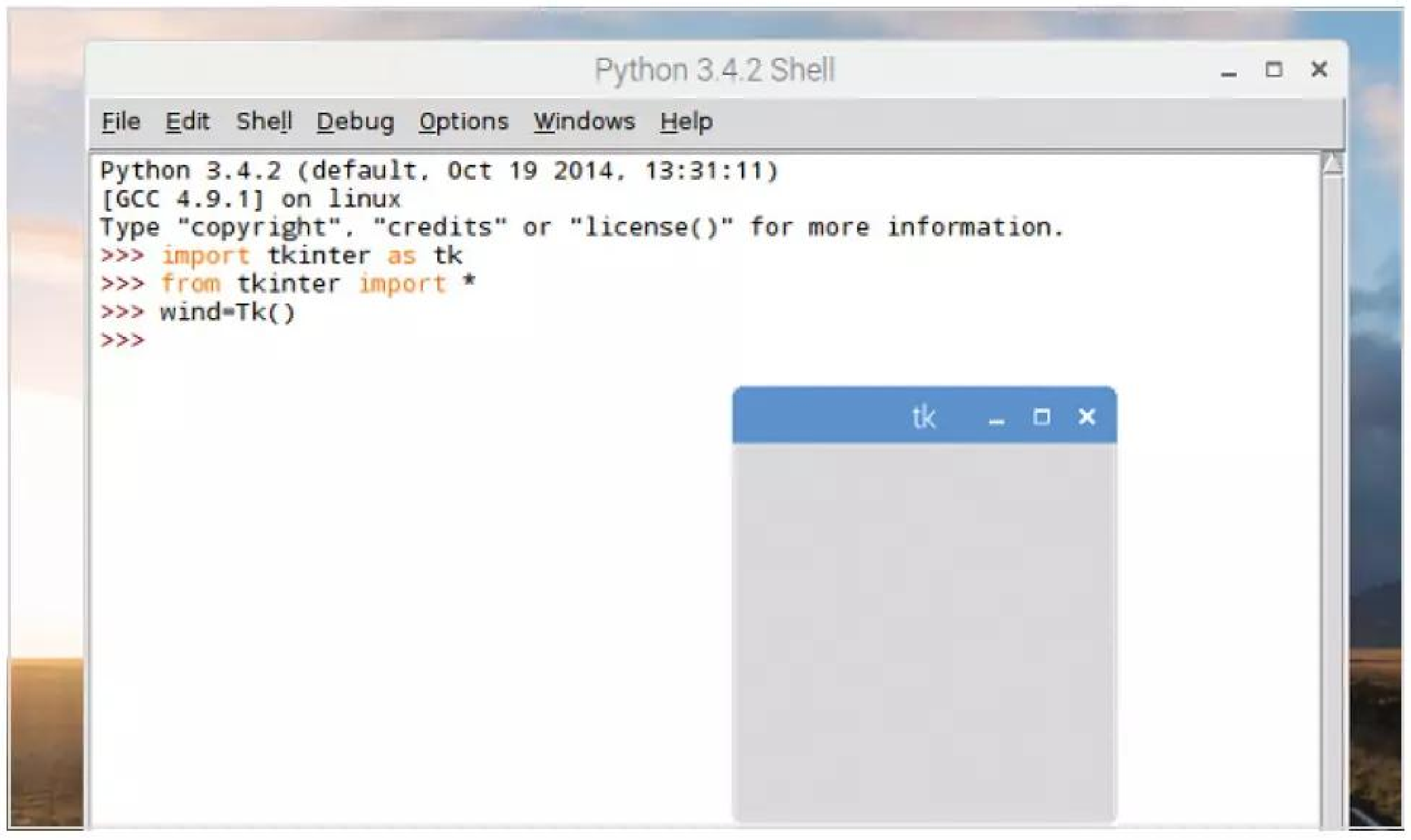

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import tkinter as tk
>>> from tkinter import *
>>> |
```

STEP 2 It's not recommended to import everything from a module using the asterisk but it won't do any harm normally. Let's begin by creating a basic GUI window, enter:

```
wind=Tk()
```

This creates a small, basic window. There's not much else to do at this point but click the X in the corner to close the window.

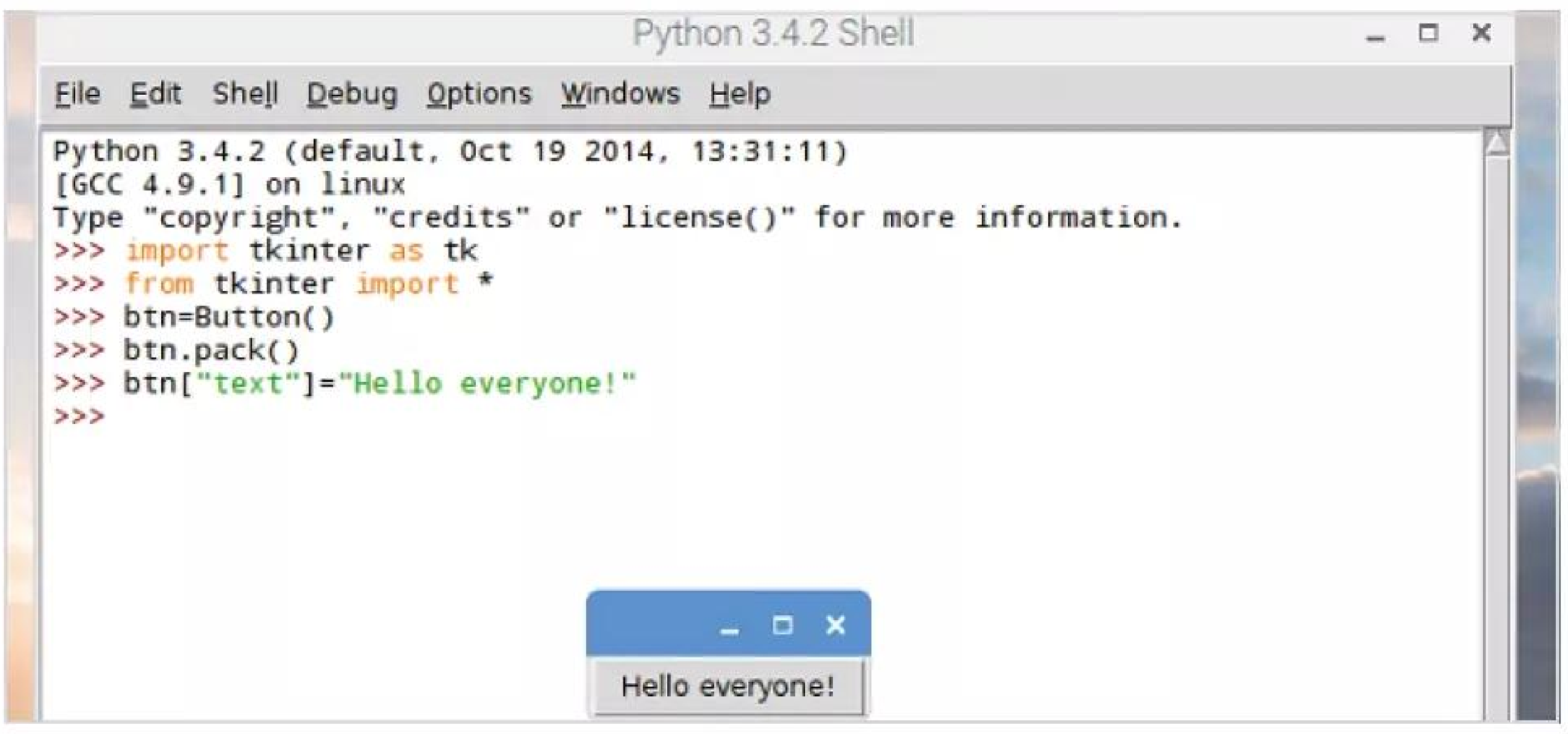

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
tk

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import tkinter as tk
>>> from tkinter import *
>>> wind=Tk()
>>>
```

STEP 3 The ideal approach is to add mainloop() into the code to control the Tkinter event loop, but we'll get to that soon. You've just created a Tkinter widget and there are several more we can play around with:

```
btn=Button()
btn.pack()
btn[“text”]=”Hello everyone!”
```

The first line focuses on the newly created window. Click back into the Shell and continue the other lines.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Hello everyone!

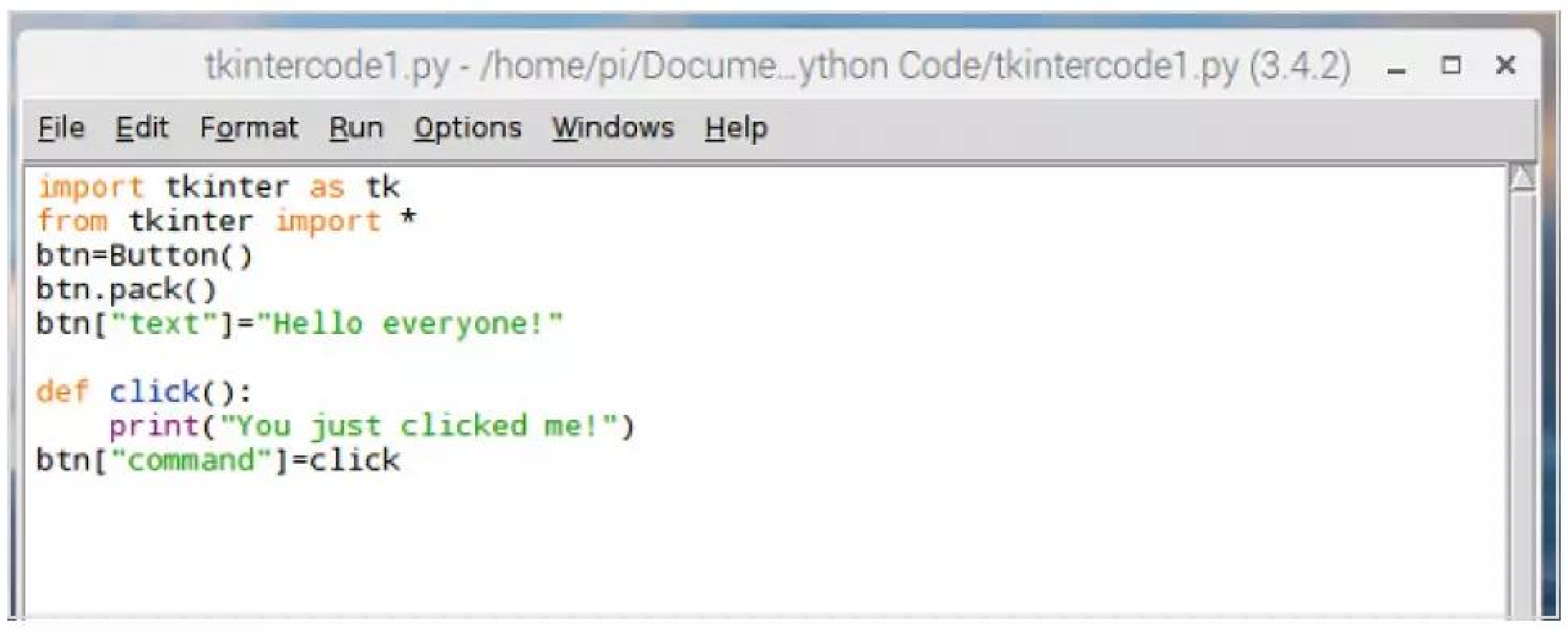
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import tkinter as tk
>>> from tkinter import *
>>> btn=Button()
>>> btn.pack()
>>> btn["text"]="Hello everyone!"
>>>
```

STEP 4 You can combine the above into a New File:

```
import tkinter as tk
from tkinter import *
btn=Button()
btn.pack()
btn[“text”]=”Hello everyone!”
```

Then add some button interactions:

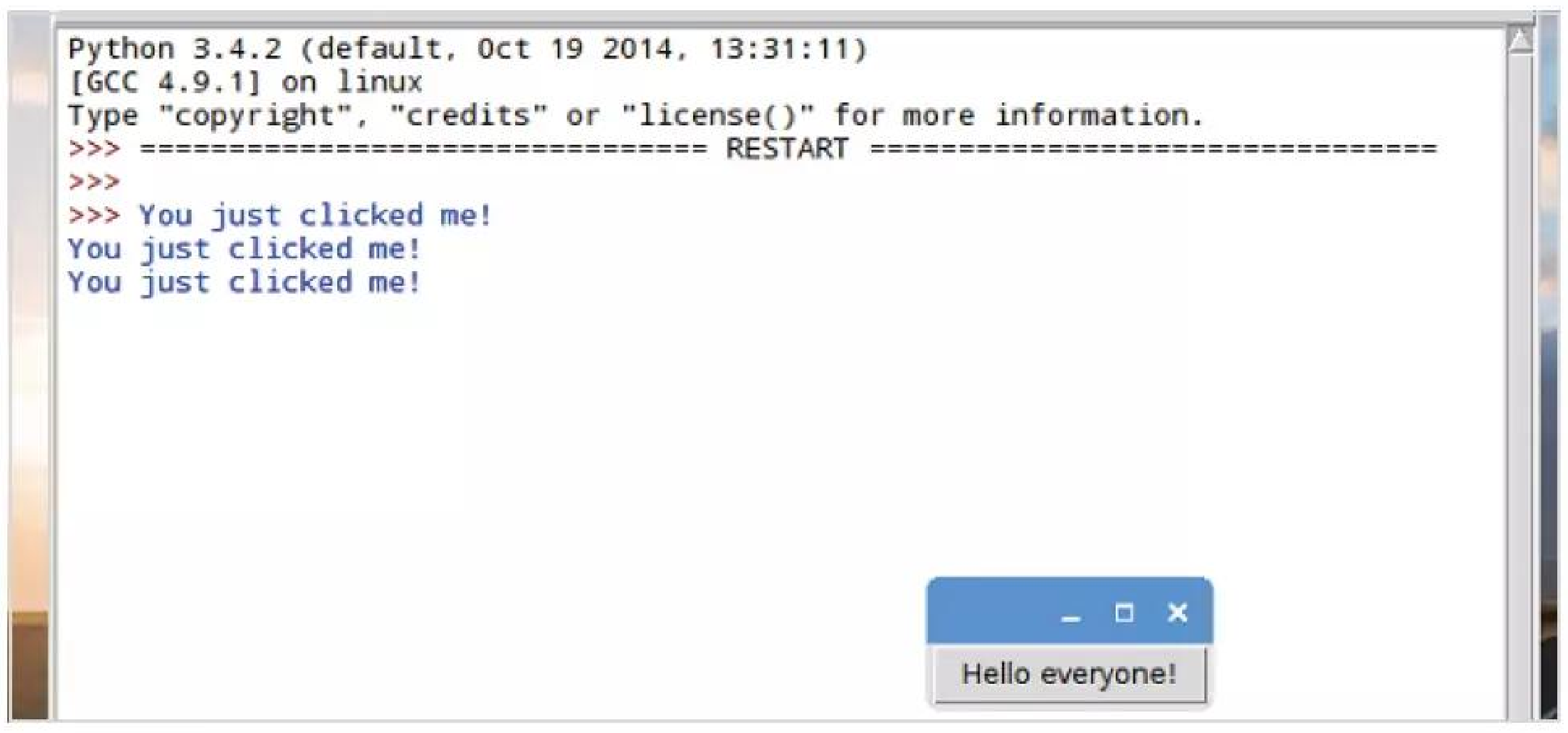
```
def click():
  print(“You just clicked me!”)
btn[“command”]=click
```


tkintercode1.py - /home/pi/Docume...ython Code/tkintercode1.py (3.4.2)
File Edit Format Run Options Windows Help

```
import tkinter as tk
from tkinter import *
btn=Button()
btn.pack()
btn["text"]="Hello everyone!"

def click():
    print("You just clicked me!")
btn["command"]=click
```

STEP 5

Save and execute the code from Step 4 and a window appears with 'Hello everyone!' inside. If you click the Hello everyone! button, the Shell will output the text 'You just clicked me!'. It's simple but shows you what can be achieved with a few lines of code.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>> You just clicked me!
You just clicked me!
You just clicked me!
```

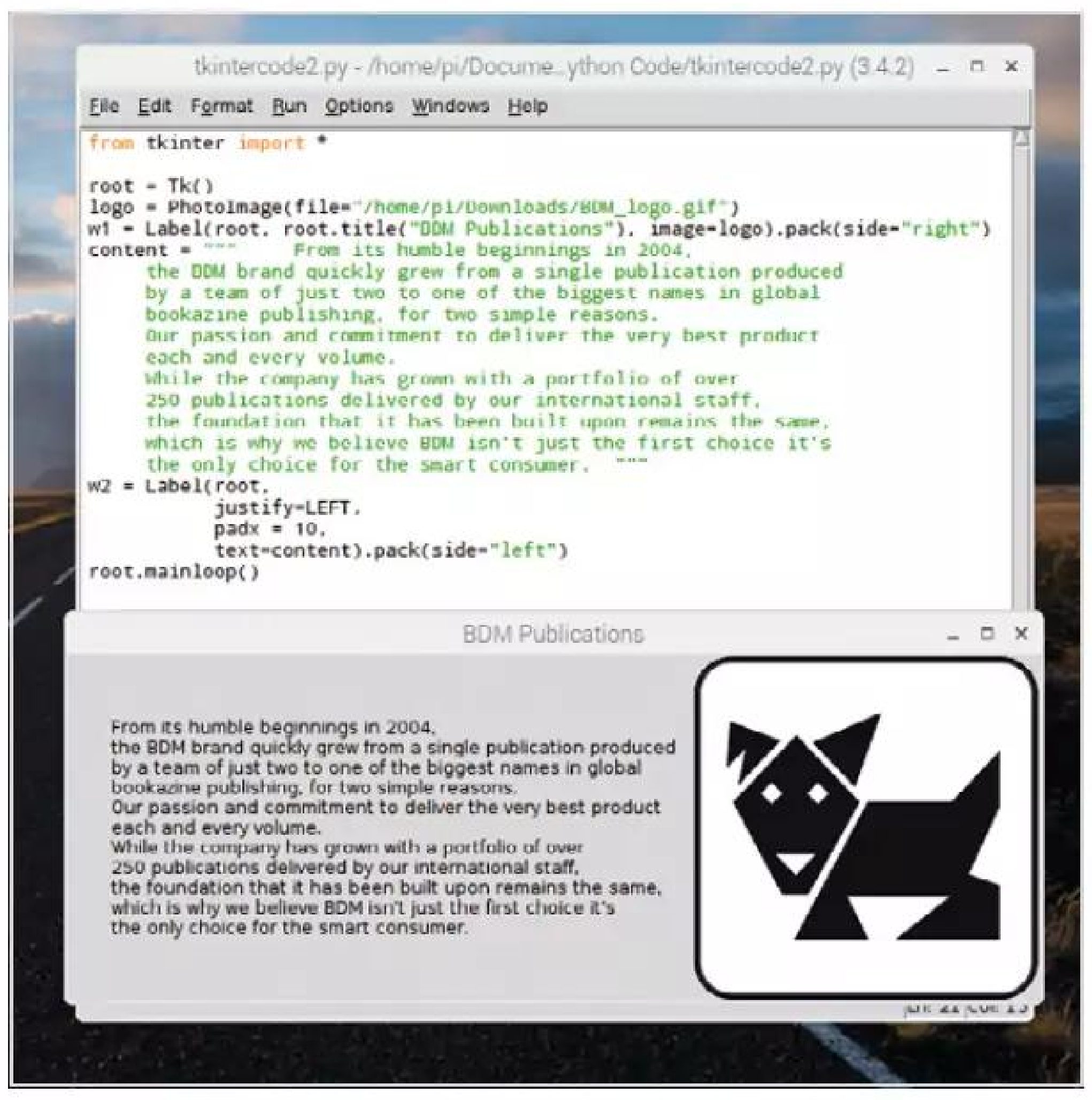
STEP 6

You can also display both text and images within a Tkinter window. However, only GIF, PGM or PPM formats are supported. So find an image and convert it before using the code. Here's an example using the BDM Publishing logo:

```
from tkinter import *

root = Tk()
logo = PhotoImage(file=”/home/pi/Downloads/BDM_logo.
gif”)
w1 = Label(root, root.title(“BDM Publications”),
image=logo).pack(side=”right”)
content = “””  From its humble beginnings in 2004,
the BDM brand quickly grew from a single publication
produced by a team of just two to one of the biggest
names in global bookazine publishing, for two simple
reasons. Our passion and commitment to deliver the
very best product each and every volume. While
the company has grown with a portfolio of over 250
publications delivered by our international staff,
the foundation that it has been built upon remains
the same, which is why we believe BDM isn’t just
the first choice it’s the only choice for the smart
consumer.  “””
w2 = Label(root,
    justify=LEFT,
    padx = 10,
    text=content).pack(side=”left”)
root.mainloop()
```

STEP 7

The previous code is quite weighty, mostly due to the content variable holding a part of BDM's About page from the company website. You can obviously change the content, the root.title and the image to suit your needs.



tkintercode2.py - /home/pi/Docume...ython Code/tkintercode2.py (3.4.2)
File Edit Format Run Options Windows Help

```
from tkinter import *

root = Tk()
logo = PhotoImage(file="/home/pi/Downloads/BDM_logo.gif")
w1 = Label(root, root.title("BDM Publications"), image=logo).pack(side="right")
content = """     From its humble beginnings in 2004,
     the BDM brand quickly grew from a single publication produced
     by a team of just two to one of the biggest names in global
     bookazine publishing, for two simple reasons.
     Our passion and commitment to deliver the very best product
     each and every volume.
     While the company has grown with a portfolio of over
     250 publications delivered by our international staff,
     the foundation that it has been built upon remains the same,
     which is why we believe BDM isn't just the first choice it's
     the only choice for the smart consumer.  """
w2 = Label(root,
           justify=LEFT,
           padx = 10,
           text=content).pack(side="left")
root.mainloop()
```

```
From its humble beginnings in 2004,
the BDM brand quickly grew from a single publication produced
by a team of just two to one of the biggest names in global
bookazine publishing, for two simple reasons.
Our passion and commitment to deliver the very best product
each and every volume.
While the company has grown with a portfolio of over
250 publications delivered by our international staff,
the foundation that it has been built upon remains the same,
which is why we believe BDM isn't just the first choice it's
the only choice for the smart consumer.
```

STEP 8

You can create radio buttons too. Try:

```
from tkinter import *
root = Tk()
v = IntVar()
Label(root, root.title(“Options”), text=”””Choose
a preferred language:”””,
  justify = LEFT, padx = 20).pack()
Radiobutton(root,
    text=”Python”,
    padx = 20,
    variable=v,
    value=1).pack(anchor=W)
Radiobutton(root,
    text=”C++”,
    padx = 20,
    variable=v,
    value=2).pack(anchor=W)
mainloop()
```

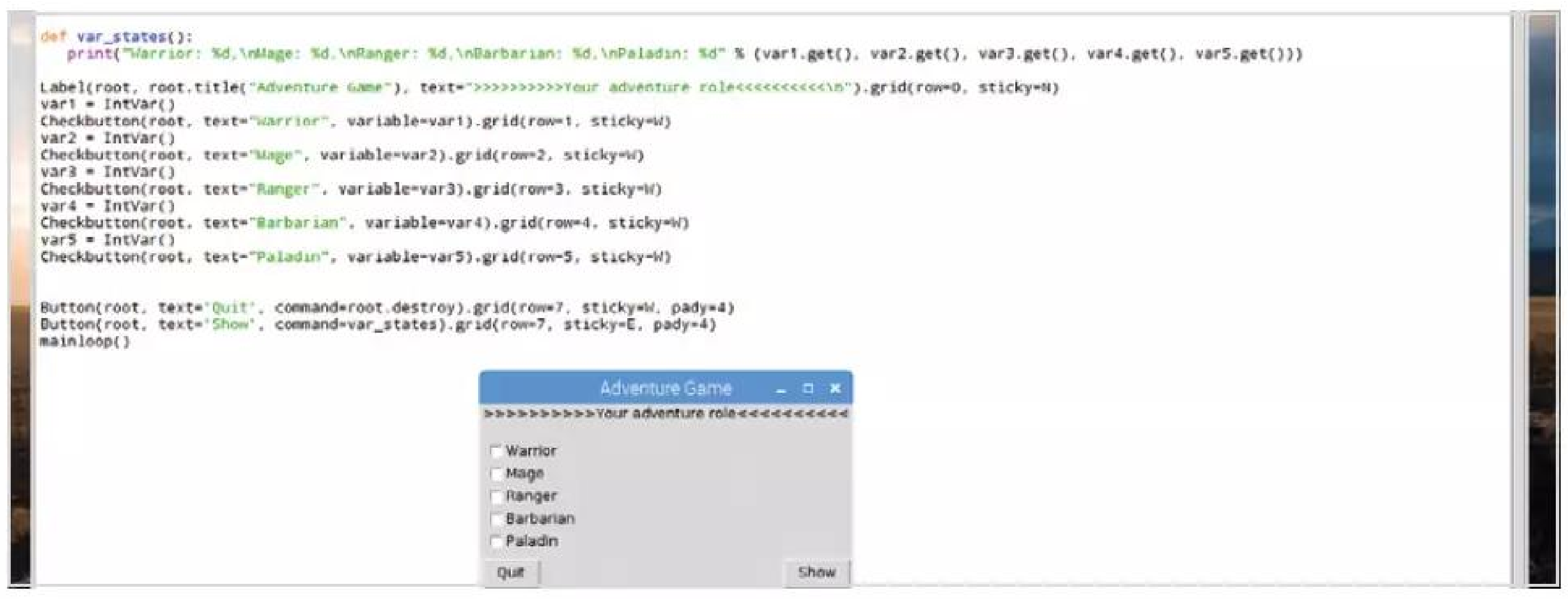
STEP 9

You can also create check boxes, with buttons and output to the Shell:

```
from tkinter import *
root = Tk()
def var_states():
  print(“Warrior: %d,\nMage: %d” % (var1.get(),
var2.get()))
Label(root, root.title(“Adventure Game”),
text=”>>>>>>>>>>Your adventure role<<<<<<<<<<”).
grid(row=0, sticky=N)
var1 = IntVar()
Checkbutton(root, text=”Warrior”, variable=var1).
grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(root, text=”Mage”, variable=var2).
grid(row=2, sticky=W)
Button(root, text=’Quit’, command=root.destroy).
grid(row=3, sticky=W, pady=4)
Button(root, text=’Show’, command=var_states).
grid(row=3, sticky=E, pady=4)
mainloop()
```

STEP 10

The code from Step 9 introduced some new geometry elements into Tkinter. Note the sticky=N, E and W arguments. These describe the locations of the check boxes and buttons (North, East, South and West). The row argument places them on separate rows. Have a play around and see what you get.



```
def var_states():
   print("Warrior: %d,\nMage: %d,\nRanger: %d,\nBarbarian: %d,\nPaladin: %d" % (var1.get(), var2.get(), var3.get(), var4.get(), var5.get()))

Label(root, root.title("Adventure Game"), text=">>>>>>>>>>Your adventure role<<<<<<<<<<\n").grid(row=0, sticky=N)
var1 = IntVar()
Checkbutton(root, text="Warrior", variable=var1).grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(root, text="Mage", variable=var2).grid(row=2, sticky=W)
var3 = IntVar()
Checkbutton(root, text="Ranger", variable=var3).grid(row=3, sticky=W)
var4 = IntVar()
Checkbutton(root, text="Barbarian", variable=var4).grid(row=4, sticky=W)
var5 = IntVar()
Checkbutton(root, text="Paladin", variable=var5).grid(row=5, sticky=W)


Button(root, text='Quit', command=root.destroy).grid(row=7, sticky=W, pady=4)
Button(root, text='Show', command=var_states).grid(row=7, sticky=E, pady=4)
mainloop()
```

Pygame Module

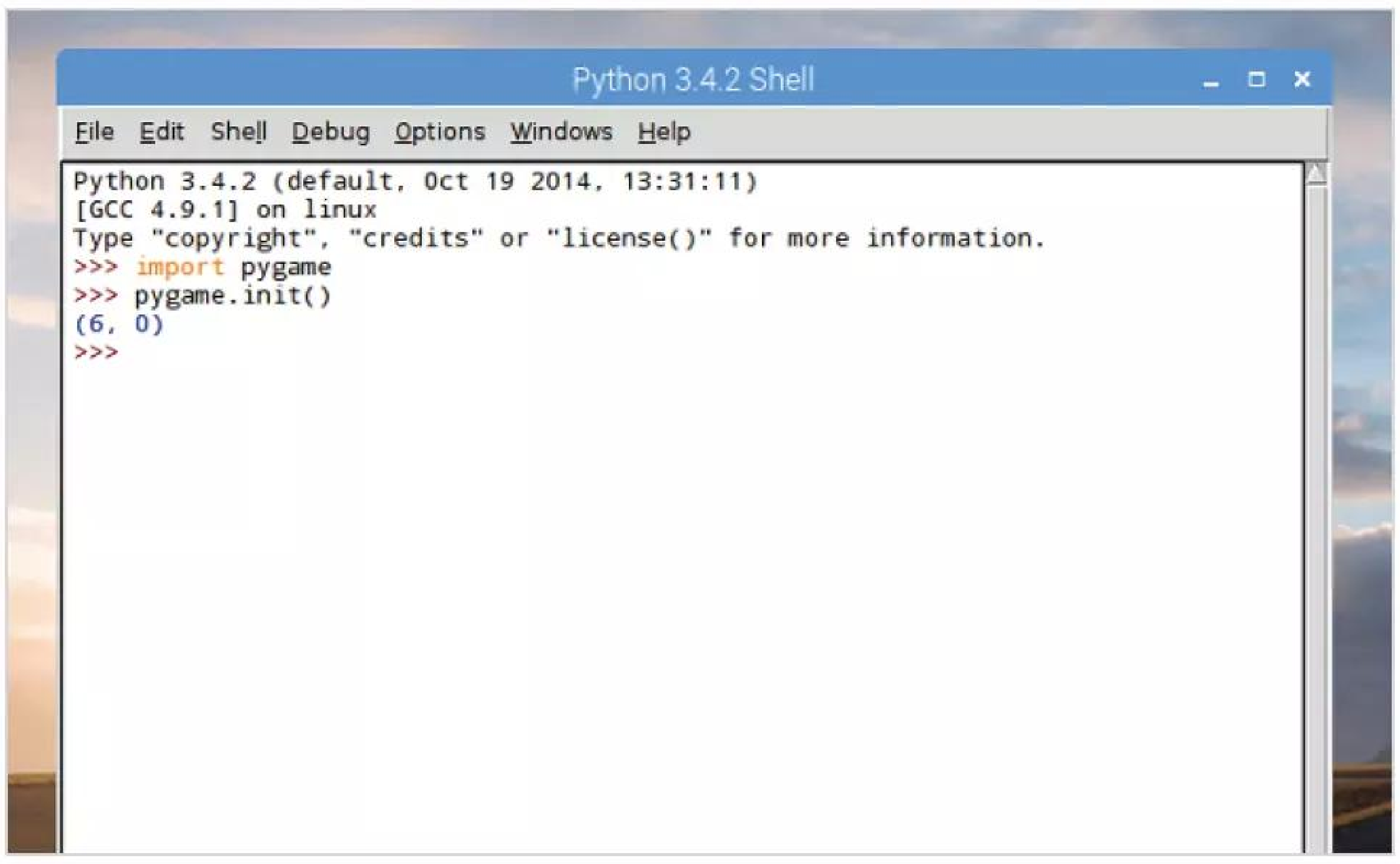
We've had a brief look at the Pygame module already but there's a lot more to it that needs exploring. Pygame was developed to help Python programmers create either graphical or text-based games.

PYGAMING

Pygame isn't an inherent module to Python but those using the Raspberry Pi will already have it installed. Everyone else will need to use: pip install pygame from the command prompt.

STEP 1 Naturally you need to load up the Pygame modules into memory before you're able to utilise them. Once that's done Pygame requires the user to initialise it prior to any of the functions being used:

```
import pygame
pygame.init()
```

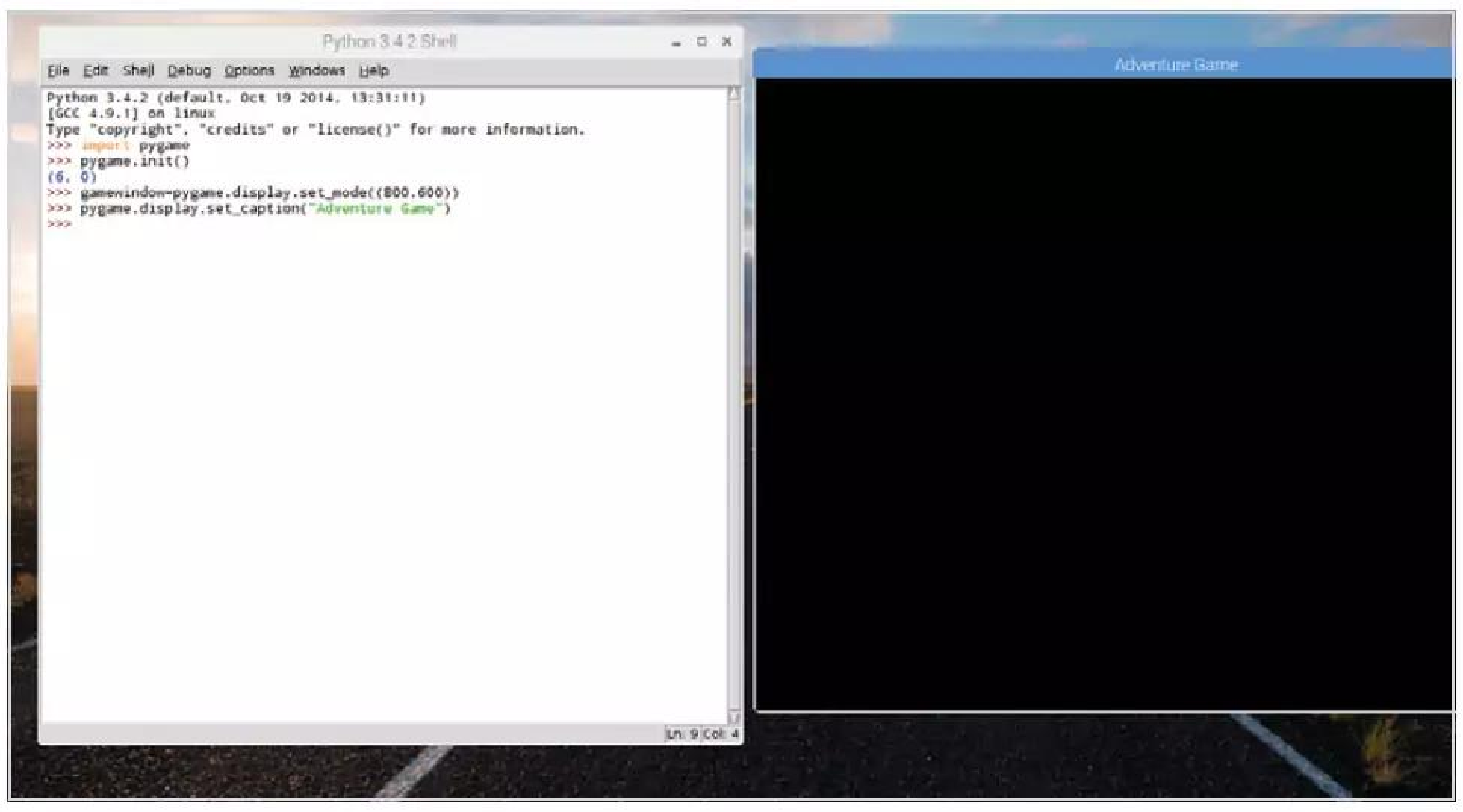


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>> pygame.init()
(6, 0)
>>>
```

STEP 2 Let's create a simple game ready window, and give it a title:

```
gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption(“Adventure Game”)
```

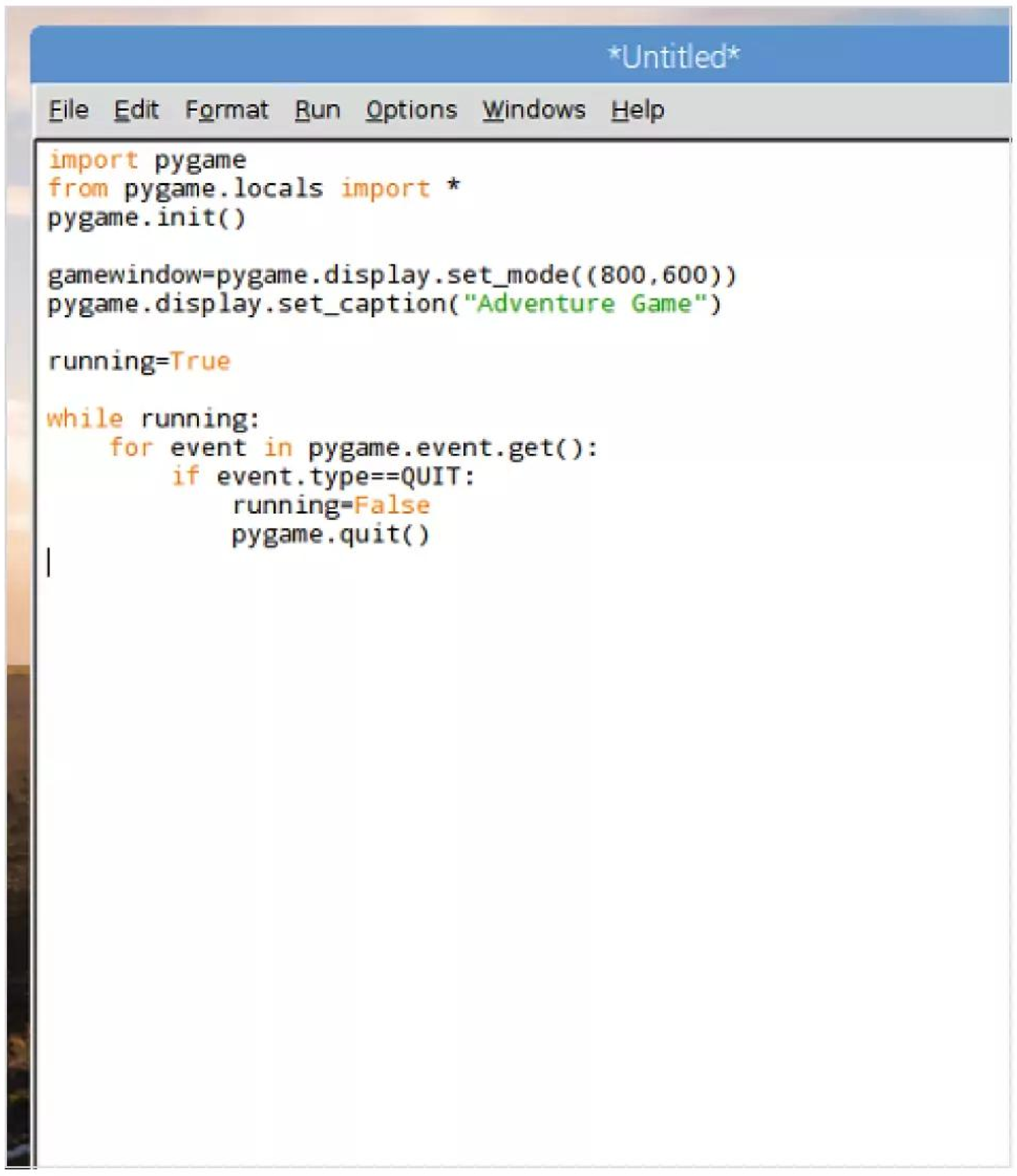
You can see that after the first line is entered, you need to click back into the IDLE Shell to continue entering code; also, you can change the title of the window to anything you like.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>> pygame.init()
(6, 0)
>>> gamewindow=pygame.display.set_mode((800,600))
>>> pygame.display.set_caption("Adventure Game")
>>>
```

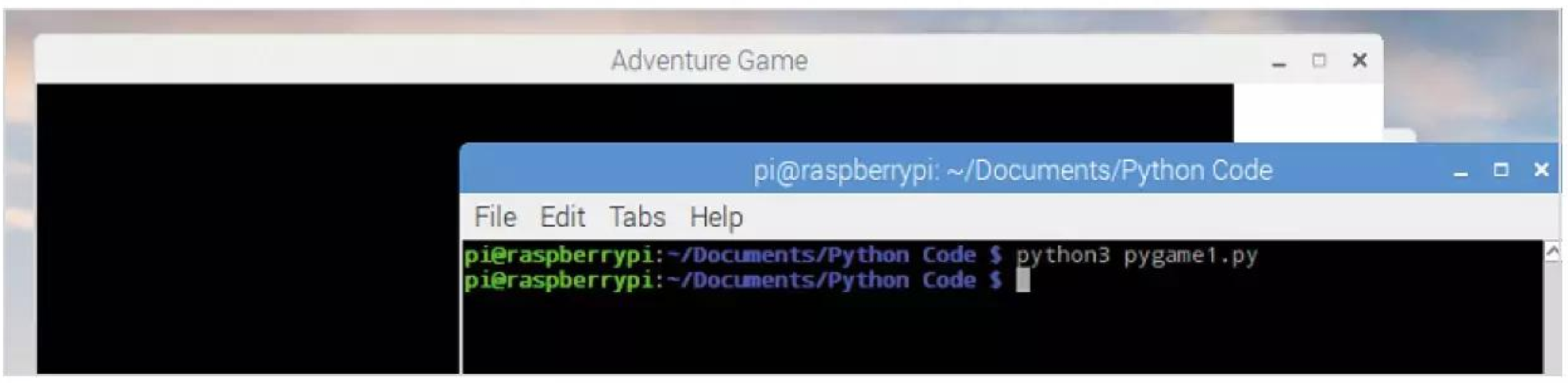
STEP 3 Sadly you can't close the newly created Pygame window without closing the Python IDLE Shell, which isn't very practical. For this reason, you need to work in the editor (New > File) and create a True/False while loop:

```
import pygame
from pygame.locals import *
pygame.init()

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption(“Adventure Game”)

running=True

while running:
 for event in pygame.event.get():
  if event.type==QUIT:
   running=False
   pygame.quit()
```



```
import pygame
from pygame.locals import *
pygame.init()

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")

running=True

while running:
    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

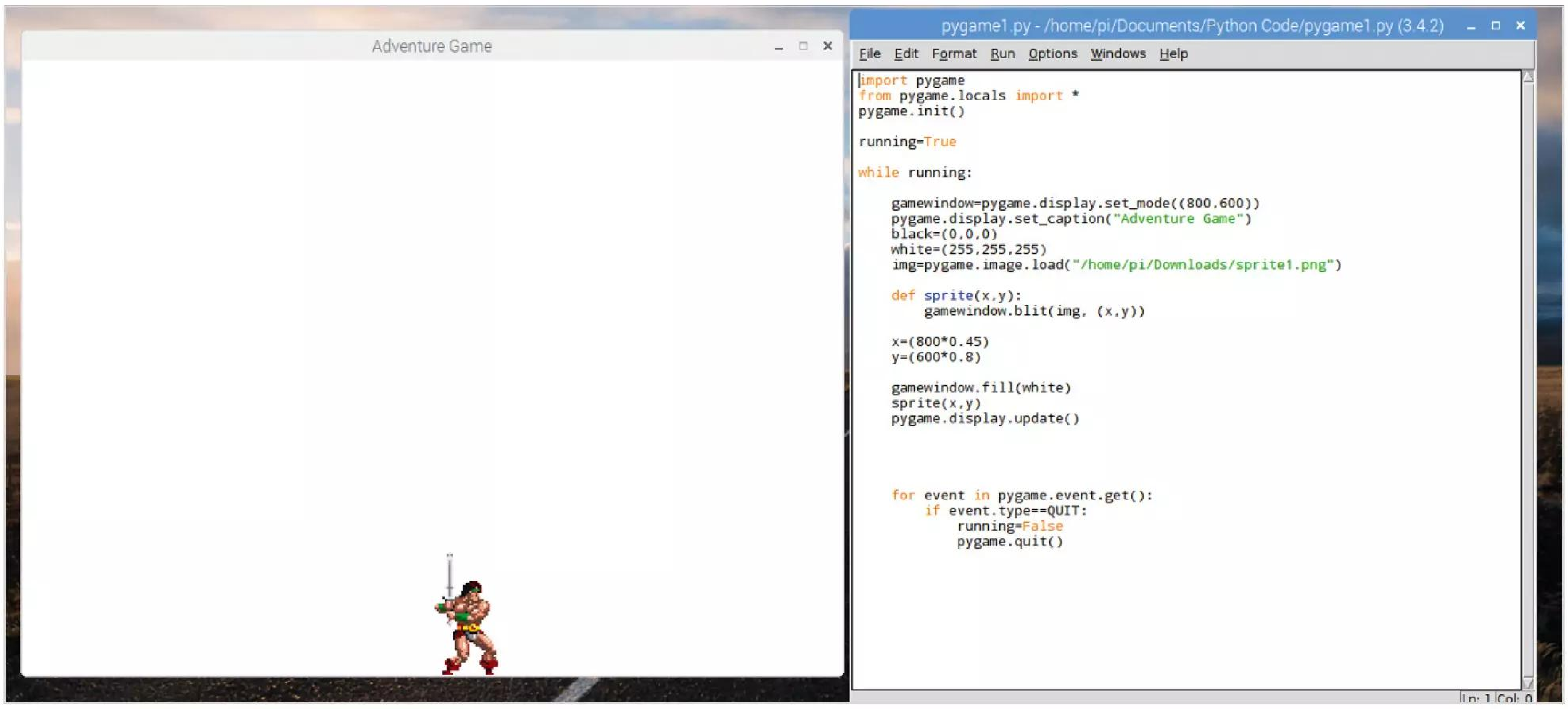
STEP 4 If the Pygame window still won’t close don’t worry, it’s just a discrepancy between the IDLE (which is written with Tkinter) and the Pygame module. If you run your code via the command line, it closes perfectly well.



```
pi@raspberrypi:~/Documents/Python Code $ python3 pygame1.py
pi@raspberrypi:~/Documents/Python Code $ 
```

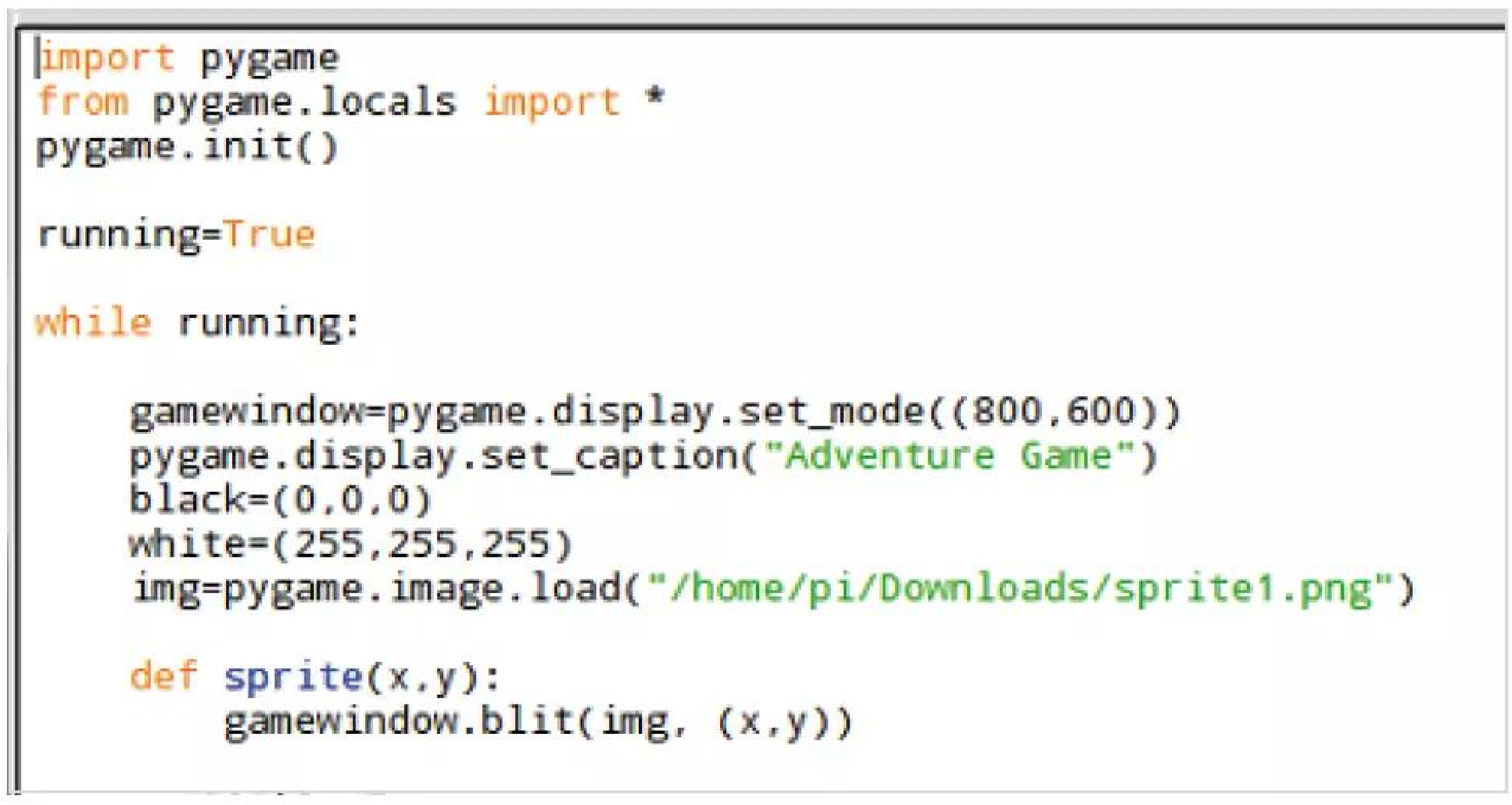
STEP 5 You’re going to shift the code around a bit now, running the main Pygame code within a while loop; it makes it neater and easier to follow. We’ve downloaded a graphic to use and we need to set some parameters for pygame:

```
import pygame
pygame.init()

running=True

while running:

  gamewindow=pygame.display.set_mode((800,600))
  pygame.display.set_caption(“Adventure Game”)
  black=(0,0,0)
  white=(255,255,255)
  img=pygame.image.load(“/home/pi/Downloads/sprite1.png”)

  def sprite(x,y):
    gamewindow.blit(img, (x,y))

  x=(800*0.45)
  y=(600*0.8)

  gamewindow.fill(white)
  sprite(x,y)
  pygame.display.update()

  for event in pygame.event.get():
    if event.type==pygame.QUIT:
      running=False
```

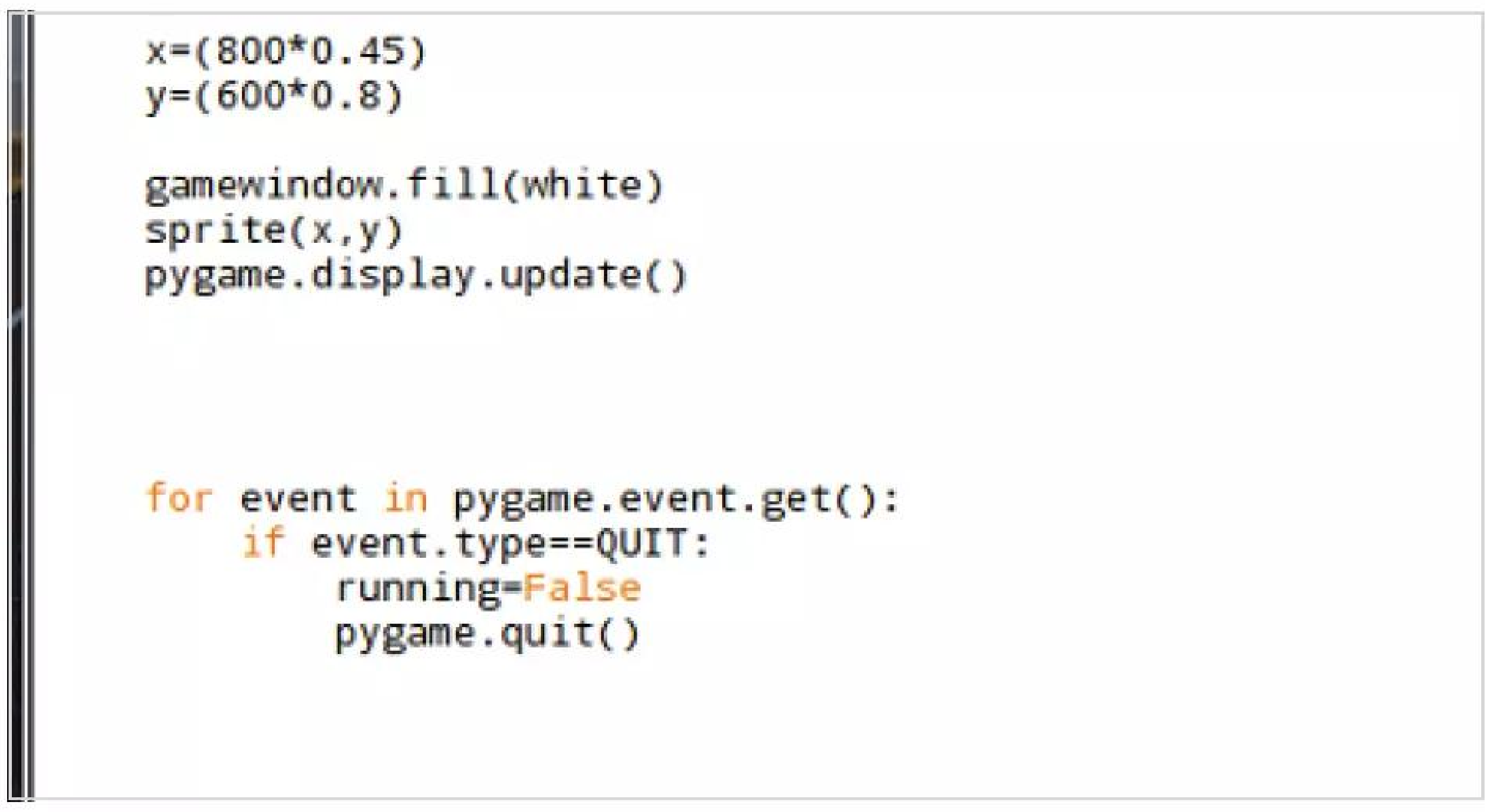

Adventure Game
pygame1.py - /home/pi/Documents/Python Code/pygame1.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
import pygame
from pygame.locals import *
pygame.init()

running=True

while running:

    gamewindow=pygame.display.set_mode((800,600))
    pygame.display.set_caption("Adventure Game")
    black=(0,0,0)
    white=(255,255,255)
    img=pygame.image.load("/home/pi/Downloads/sprite1.png")

    def sprite(x,y):
        gamewindow.blit(img, (x,y))

    x=(800*0.45)
    y=(600*0.8)

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()



    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

STEP 6 Let’s quickly go through the code changes. We’ve defined two colours, black and white together with their respective RGB colour values. Next we’ve loaded the downloaded image called sprite1.png and allocated it to the variable img; and also defined a sprite function and the Blit function will allow us to eventually move the image.



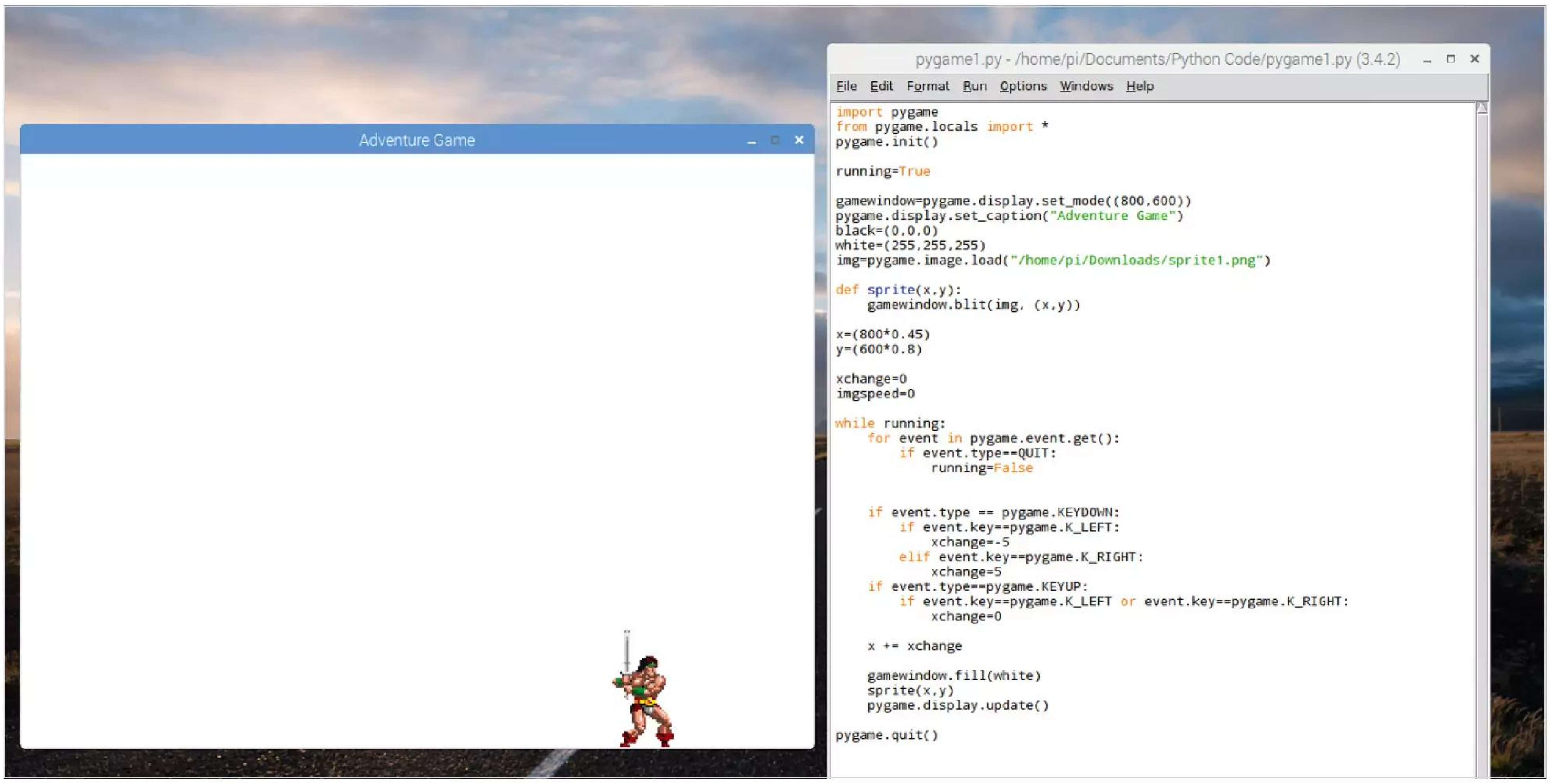
```
import pygame
from pygame.locals import *
pygame.init()

running=True

while running:

    gamewindow=pygame.display.set_mode((800,600))
    pygame.display.set_caption("Adventure Game")
    black=(0,0,0)
    white=(255,255,255)
    img=pygame.image.load("/home/pi/Downloads/sprite1.png")

    def sprite(x,y):
        gamewindow.blit(img, (x,y))
```



```
    x=(800*0.45)
    y=(600*0.8)

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()



    for event in pygame.event.get():
        if event.type==QUIT:
            running=False
            pygame.quit()
```

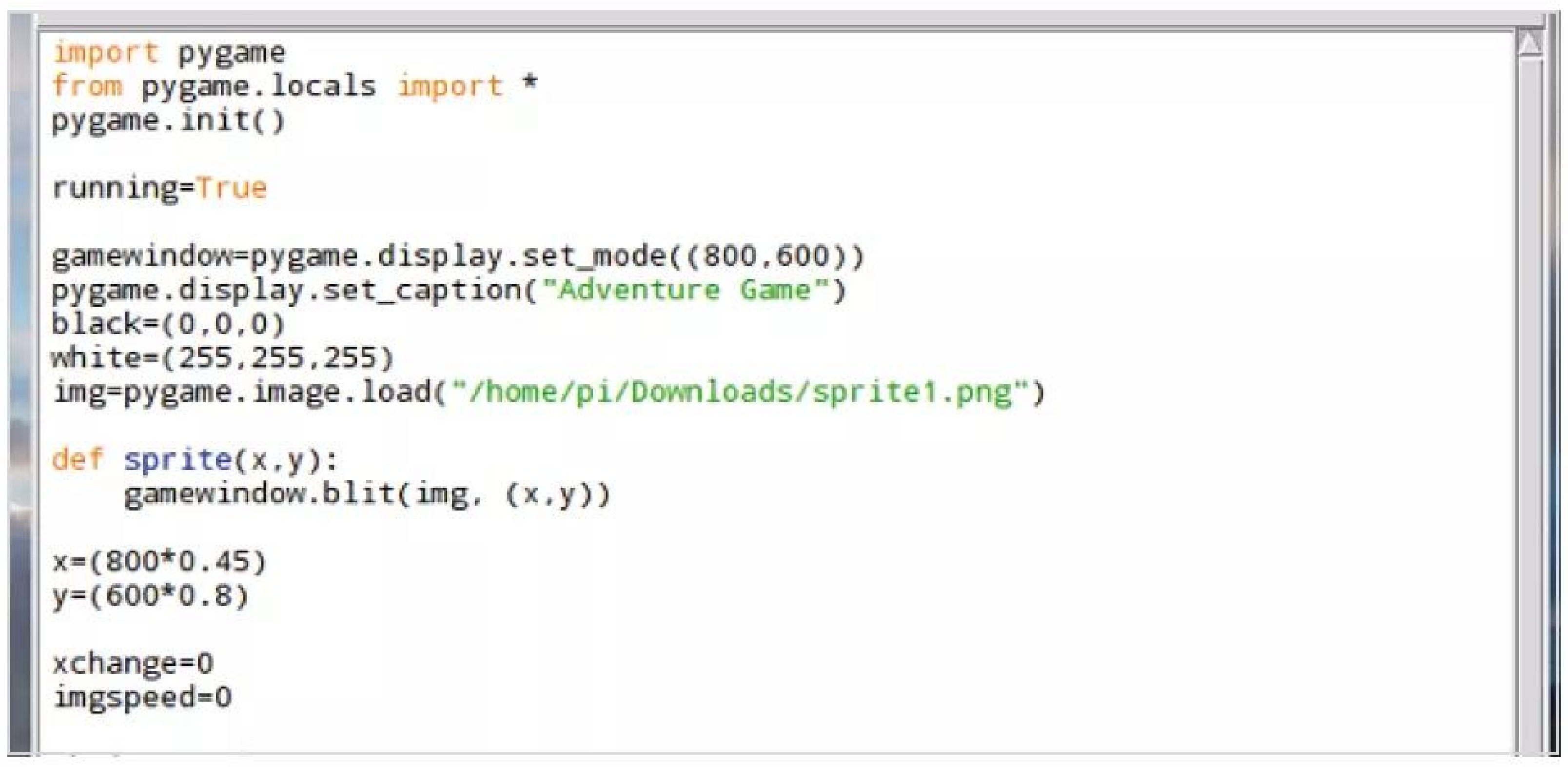
STEP 7 Now we can change the code around again, this time containing a movement option within the while loop, and adding the variables needed to move the sprite around the screen:

```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption(“Adventure Game”)
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load(“/home/pi/Downloads/sprite1.
png”)

def sprite(x,y):
  gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0

while running:
  for event in pygame.event.get():
    if event.type==QUIT:
      running=False

  if event.type == pygame.KEYDOWN:
    if event.key==pygame.K_LEFT:
      xchange=-5
    elif event.key==pygame.K_RIGHT:
      xchange=5
  if event.type==pygame.KEYUP:
    if event.key==pygame.K_LEFT or event
key==pygame.K_RIGHT:
      xchange=0

  x += xchange

  gamewindow.fill(white)
  sprite(x,y)
  pygame.display.update()

pygame.quit()
```

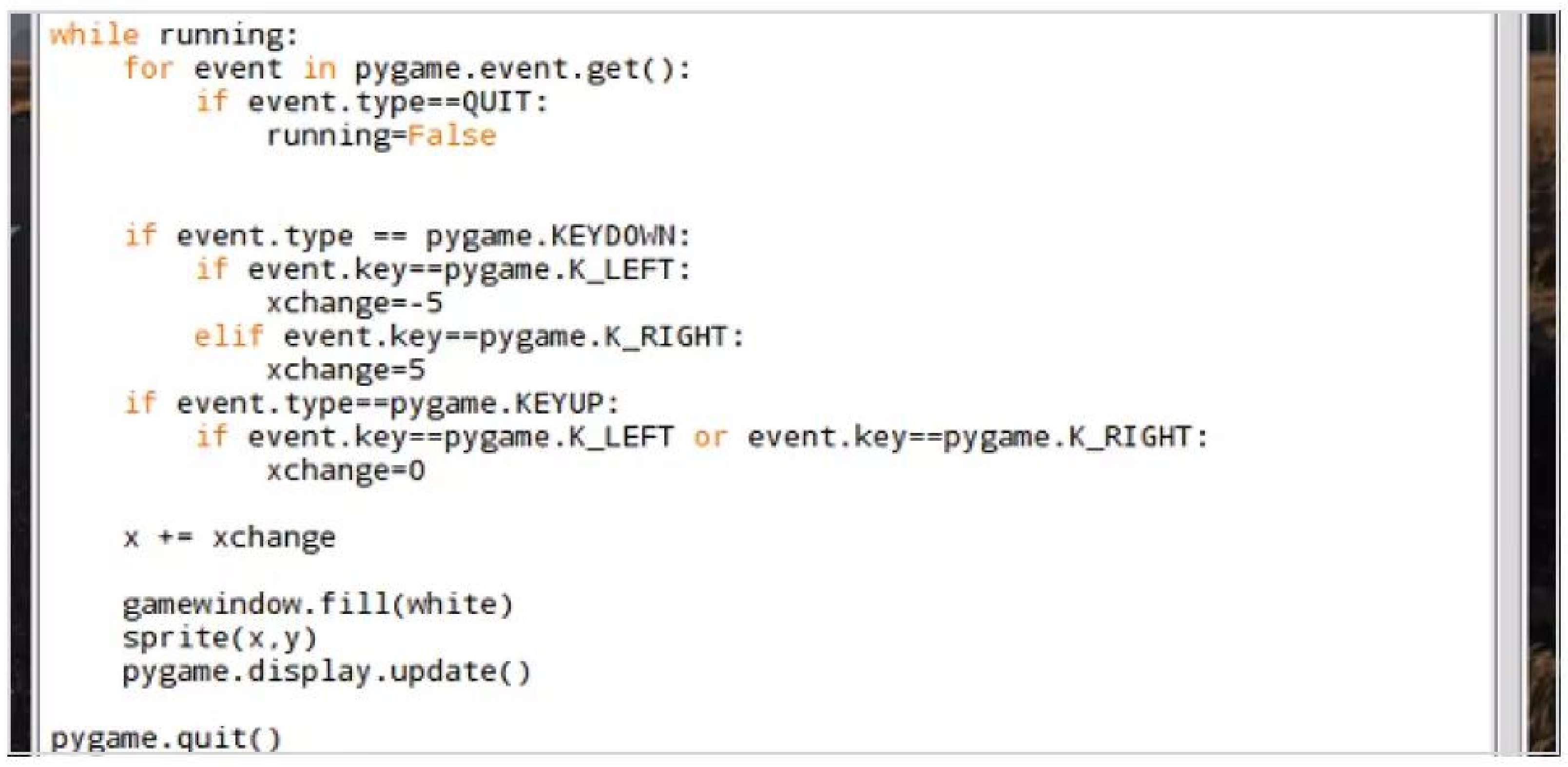

Adventure Game
pygame1.py - /home/pi/Documents/Python Code/pygame1.py (3.4.2)
File Edit Format Run Options Windows Help

```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/sprite1.png")

def sprite(x,y):
    gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0

while running:
    for event in pygame.event.get():
        if event.type==QUIT:
            running=False


    if event.type == pygame.KEYDOWN:
        if event.key==pygame.K_LEFT:
            xchange=-5
        elif event.key==pygame.K_RIGHT:
            xchange=5
    if event.type==pygame.KEYUP:
        if event.key==pygame.K_LEFT or event.key==pygame.K_RIGHT:
            xchange=0

    x += xchange

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()

pygame.quit()
```

STEP 8 Copy the code down and using the left and right arrow keys on the keyboard you can move your sprite across the bottom of the screen. Now, it looks like you have the makings of a classic arcade 2D scroller in the works.



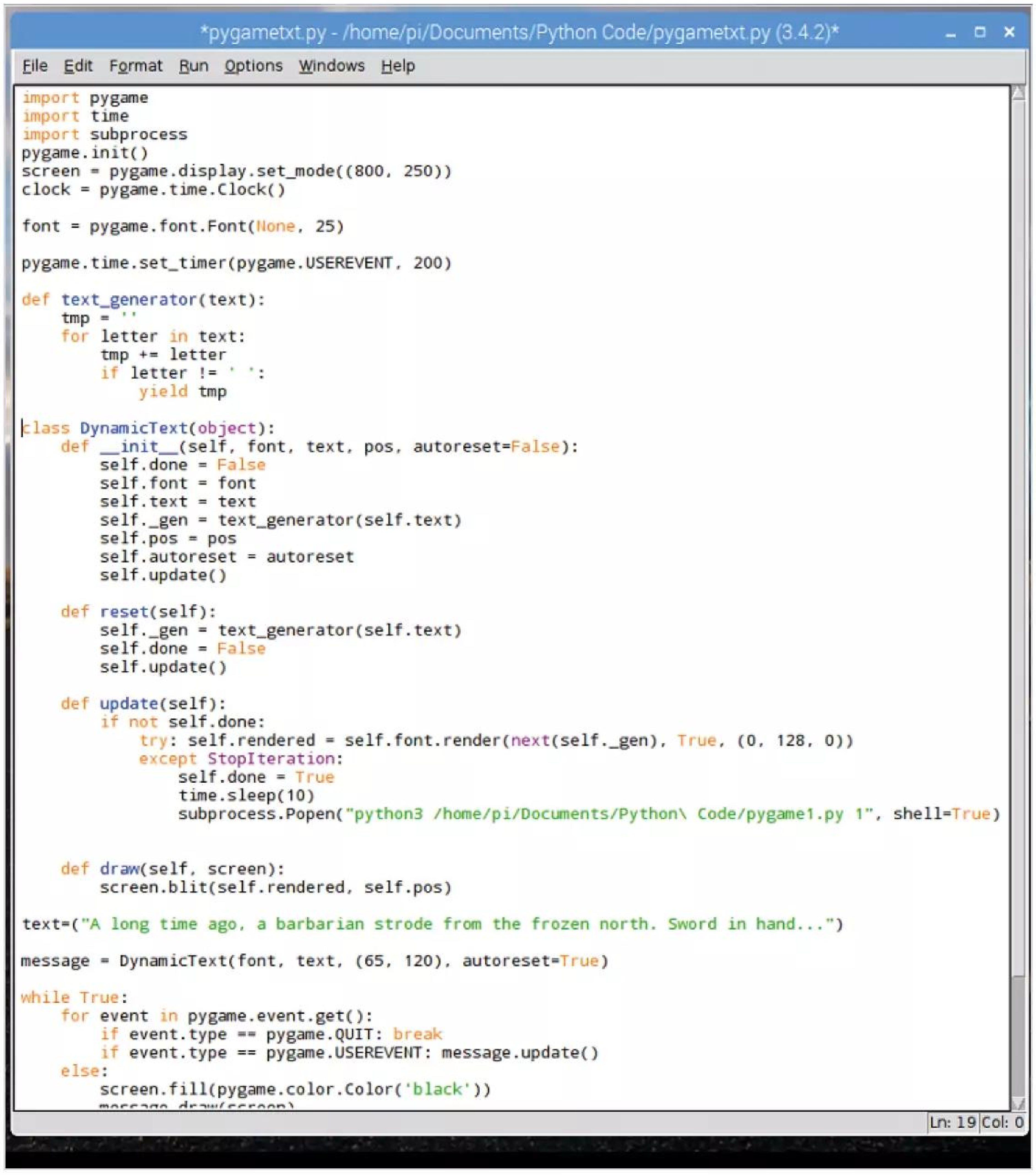
```
import pygame
from pygame.locals import *
pygame.init()

running=True

gamewindow=pygame.display.set_mode((800,600))
pygame.display.set_caption("Adventure Game")
black=(0,0,0)
white=(255,255,255)
img=pygame.image.load("/home/pi/Downloads/sprite1.png")

def sprite(x,y):
    gamewindow.blit(img, (x,y))

x=(800*0.45)
y=(600*0.8)

xchange=0
imgspeed=0
```



```
while running:
    for event in pygame.event.get():
        if event.type==QUIT:
            running=False


    if event.type == pygame.KEYDOWN:
        if event.key==pygame.K_LEFT:
            xchange=-5
        elif event.key==pygame.K_RIGHT:
            xchange=5
    if event.type==pygame.KEYUP:
        if event.key==pygame.K_LEFT or event.key==pygame.K_RIGHT:
            xchange=0

    x += xchange

    gamewindow.fill(white)
    sprite(x,y)
    pygame.display.update()

pygame.quit()
```

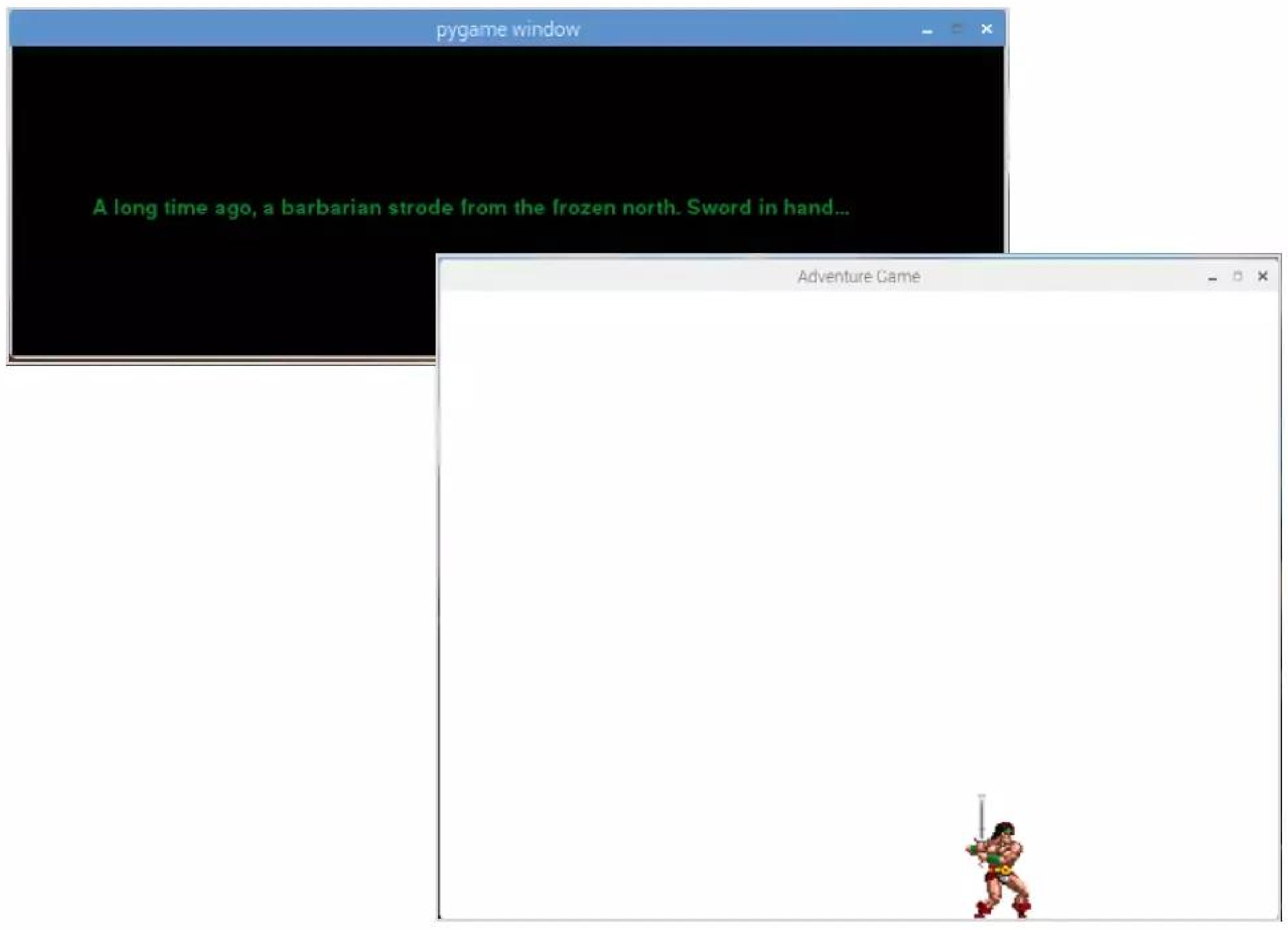
STEP 9 You can now implement a few additions and utilise some previous tutorial code. The new elements are the subprocess module, of which one function allows us to launch a second Python script from within another; and we're going to create a New File called pygametxt.py:

```
import pygame
import time
import subprocess
pygame.init()
screen = pygame.display.set_mode((800, 250))
clock = pygame.time.Clock()

font = pygame.font.Font(None, 25)

pygame.time.set_timer(pygame.USEREVENT, 200)

def text_generator(text):
  tmp = ‘’
  for letter in text:
    tmp += letter
    if letter != ‘ ‘:
      yield tmp

class DynamicText(object):
  def __init__(self, font, text, pos,
autoreset=False):
    self.done = False
    self.font = font
    self.text = text
    self._gen = text_generator(self.text)
    self.pos = pos
    self.autoreset = autoreset
    self.update()

  def reset(self):
    self._gen = text_generator(self.text)
    self.done = False
    self.update()

  def update(self):
    if not self.done:
      try: self.rendered = self.font.
render(next(self._gen), True, (0, 128, 0))
      except StopIteration:
        self.done = True
        time.sleep(10)
        subprocess.Popen(“python3 /home/pi/Documents/
Python\ Code/pygame1.py 1”, shell=True)

  def draw(self, screen):
    screen.blit(self.rendered, self.pos)

text=(“A long time ago, a barbarian strode from the
frozen north. Sword in hand...”)

message = DynamicText(font, text, (65, 120),
autoreset=True)

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT: break
    if event.type == pygame.USEREVENT: message.
update()
  else:
    screen.fill(pygame.color.Color(‘black’))
    message.draw(screen)
    pygame.display.flip()
    clock.tick(60)
    continue
  break

pygame.quit()
```


pygametxt.py - /home/pi/Documents/Python Code/pygametxt.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 19 Col: 0

```
import pygame
import time
import subprocess
pygame.init()
screen = pygame.display.set_mode((800, 250))
clock = pygame.time.Clock()

font = pygame.font.Font(None, 25)

pygame.time.set_timer(pygame.USEREVENT, 200)

def text_generator(text):
    tmp = ''
    for letter in text:
        tmp += letter
        if letter != ' ':
            yield tmp

class DynamicText(object):
    def __init__(self, font, text, pos, autoreset=False):
        self.done = False
        self.font = font
        self.text = text
        self._gen = text_generator(self.text)
        self.pos = pos
        self.autoreset = autoreset
        self.update()

    def reset(self):
        self._gen = text_generator(self.text)
        self.done = False
        self.update()

    def update(self):
        if not self.done:
            try: self.rendered = self.font.render(next(self._gen), True, (0, 128, 0))
            except StopIteration:
                self.done = True
                time.sleep(10)
                subprocess.Popen("python3 /home/pi/Documents/Python\ Code/pygame1.py 1", shell=True)


    def draw(self, screen):
        screen.blit(self.rendered, self.pos)

text=("A long time ago, a barbarian strode from the frozen north. Sword in hand...")

message = DynamicText(font, text, (65, 120), autoreset=True)

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: break
        if event.type == pygame.USEREVENT: message.update()
    else:
        screen.fill(pygame.color.Color('black'))
```

STEP 10 When you run this code it will display a long, narrow Pygame window with the intro text scrolling to the right. After a pause of ten seconds, it then launches the main game Python script where you can move the warrior sprite around. Overall the effect is quite good but there's always room for improvement.



```
A long time ago, a barbarian strode from the frozen north. Sword in hand...
```

Create Your Own Modules

Large programs can be much easier to manage if you break them up into smaller parts and import the parts you need as modules. Learning to build your own modules also makes it easier to understand how they work.

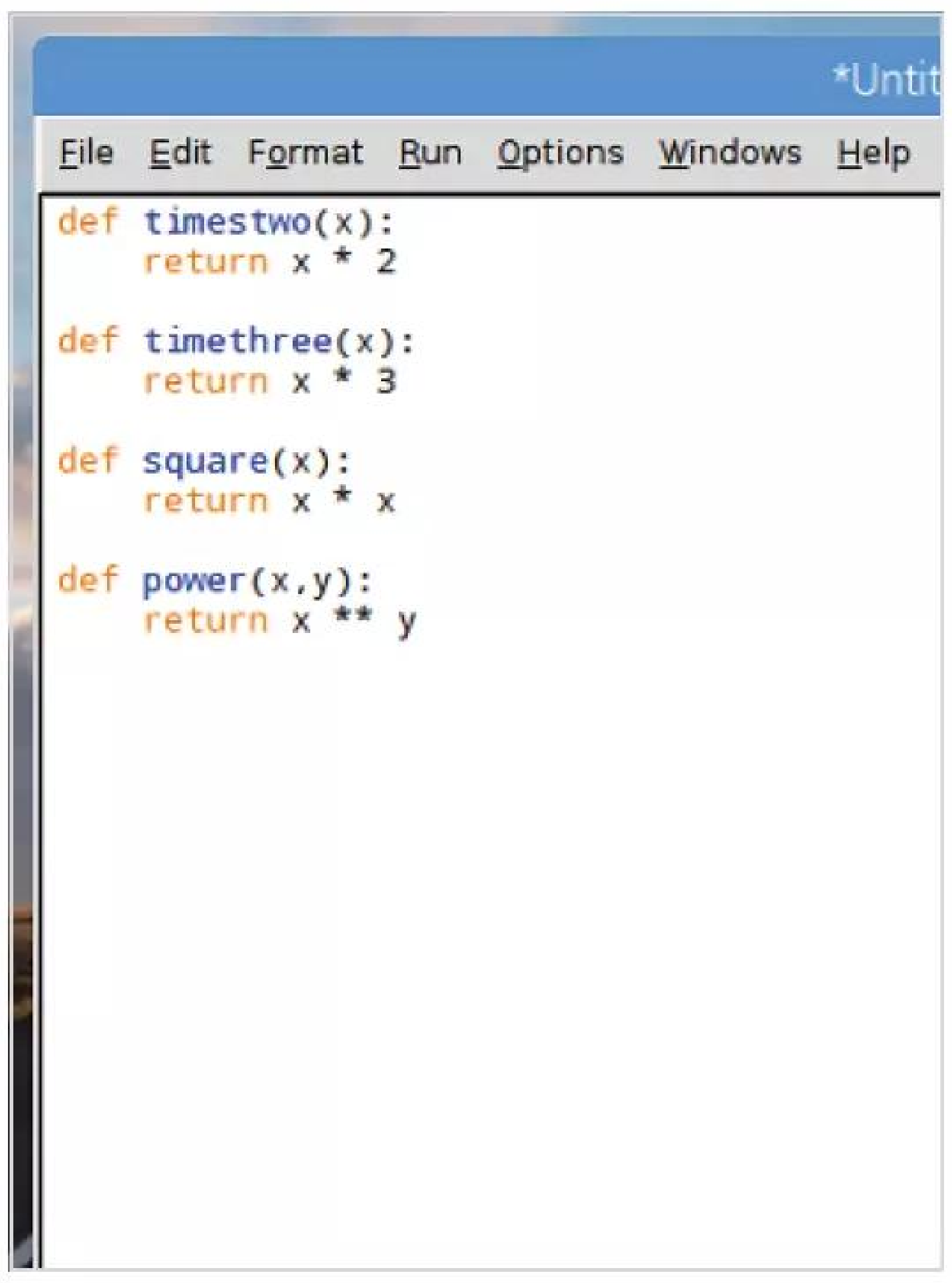
BUILDING MODULES

Modules are Python files, containing code, that you save using a .py extension. These are then imported into Python using the now familiar import command.

STEP 1

Let's start by creating a set of basic mathematics functions. Multiply a number by two, three and square or raise a number to an exponent (power). Create a New File in the IDLE and enter:

```
def timestwo(x):
        return x * 2

def timesthree(x):
        return x * 3

def square(x):
        return x * x

def power(x,y):
        return x ** y
```


*Untit
File Edit Format Run Options Windows Help

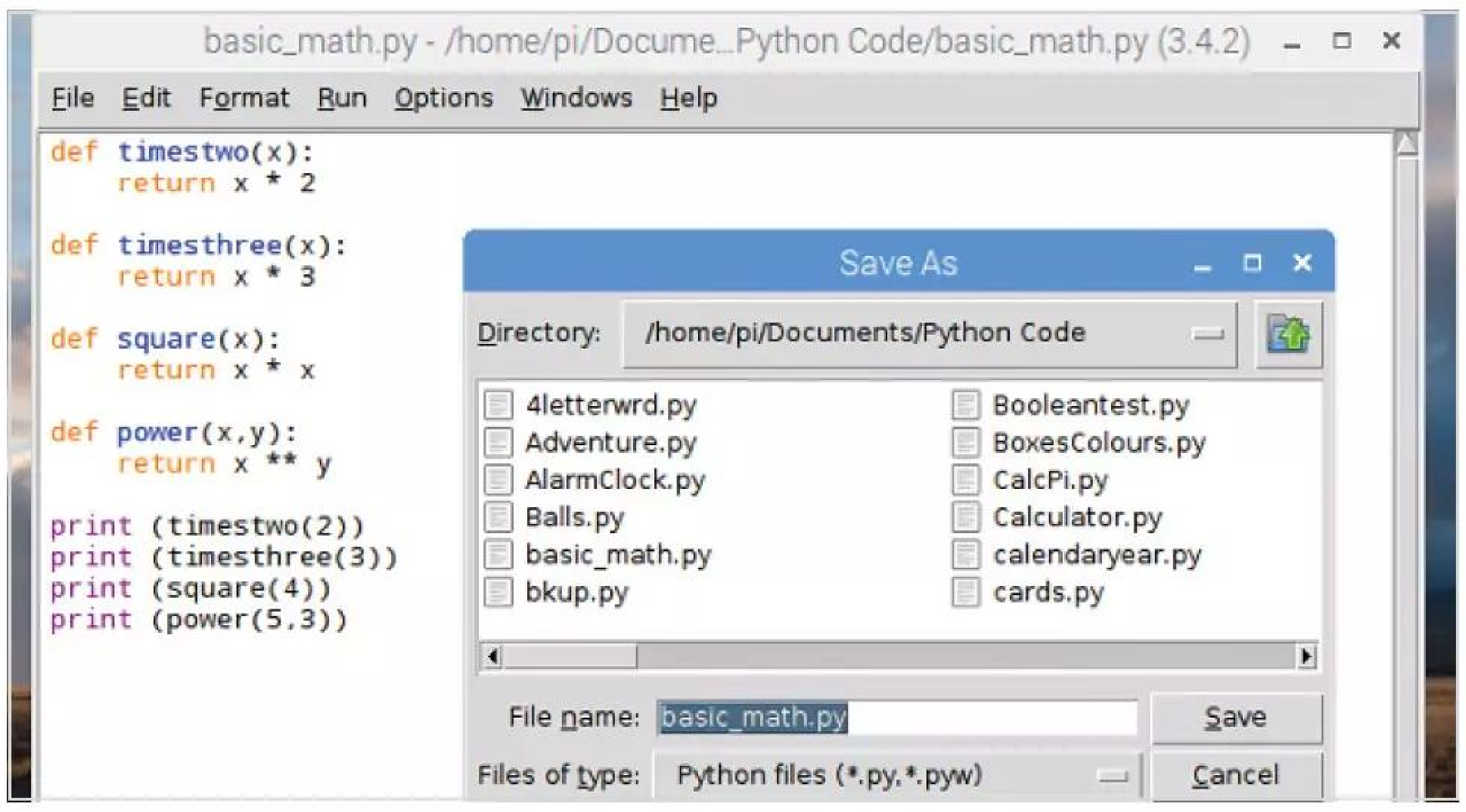
```
def timestwo(x):
    return x * 2

def timethree(x):
    return x * 3

def square(x):
    return x * x

def power(x,y):
    return x ** y
```

STEP 2

Under the above code, enter functions to call the code:

```
print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

Save the program as basic_math.py and execute it to get the results.


basic_math.py - /home/pi/Docume...Python Code/basic_math.py (3.4.2)
File Edit Format Run Options Windows Help
Save As
4letterwrd.py
Adventure.py
AlarmClock.py
Balls.py
basic_math.py
bkup.py
Booleantest.py
BoxesColours.py
CalcPi.py
Calculator.py
calendaryear.py
cards.py
Save
Cancel

```
def timestwo(x):
    return x * 2

def timesthree(x):
    return x * 3

def square(x):
    return x * x

def power(x,y):
    return x ** y

print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

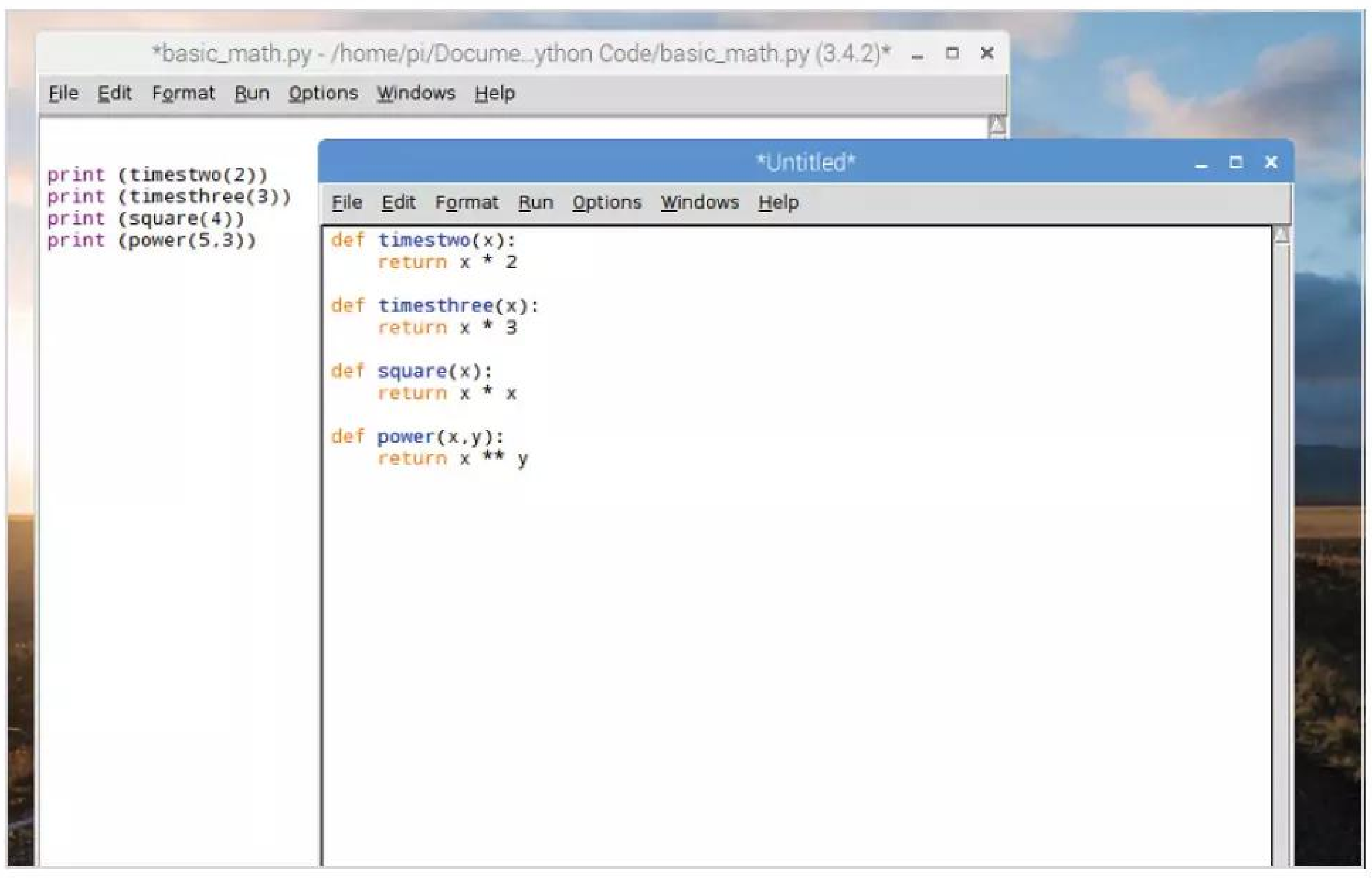
Directory: /home/pi/Documents/Python Code

File name: basic_math.py

Files of type: Python files (*.py,*.pyw)

STEP 3

Now you're going to take the function definitions out of the program and into a separate file. Highlight the function definitions and choose Edit > Cut. Choose File > New File and use Edit > Paste in the new window. You now have two separate files, one with the function definitions, the other with the function calls.

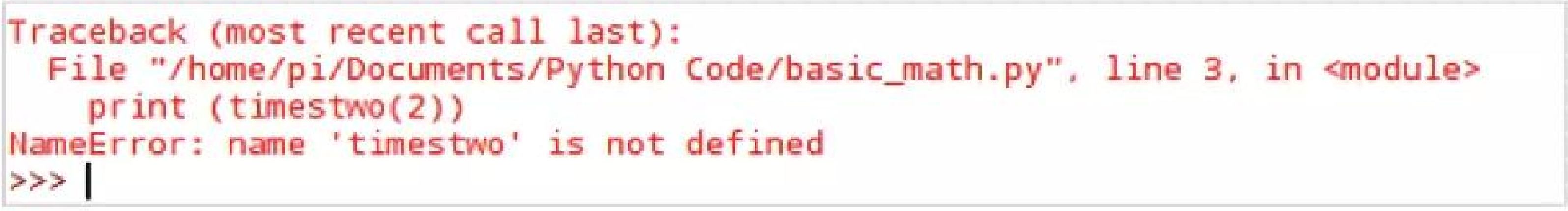

basic_math.py - /home/pi/Docume...ython Code/basic_math.py (3.4.2)
File Edit Format Run Options Windows Help
Untitled
File Edit Format Run Options Windows Help

```
print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

```
def timestwo(x):
    return x * 2

def timesthree(x):
    return x * 3

def square(x):
    return x * x

def power(x,y):
    return x ** y
```

STEP 4

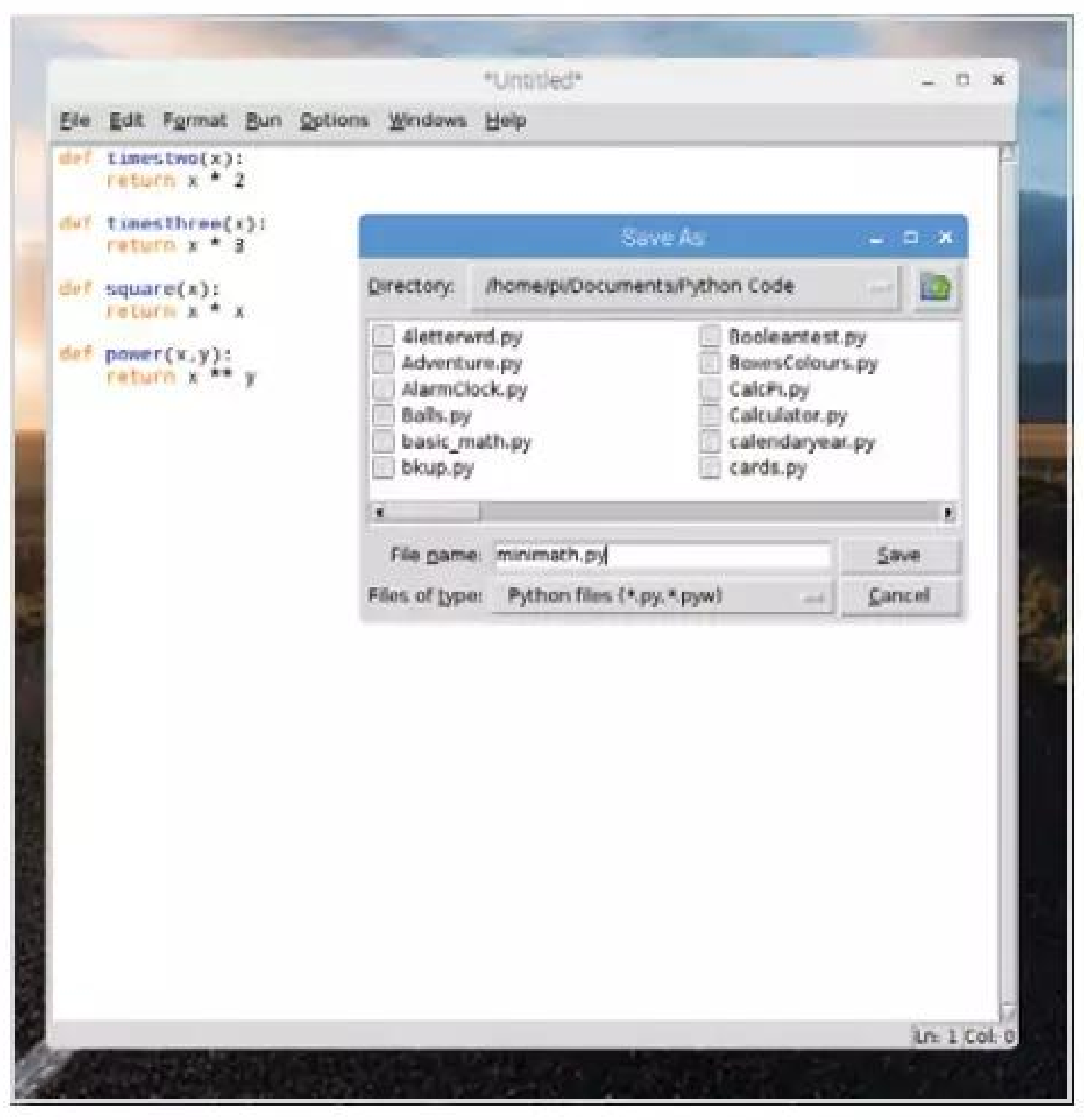
If you now try and execute the basic_math.py code again, the error **'NameError: name 'timestwo' is not defined'** will be displayed. This is due to the code no longer having access to the function definitions.



```
Traceback (most recent call last):
  File "/home/pi/Documents/Python Code/basic_math.py", line 3, in <module>
    print (timestwo(2))
NameError: name 'timestwo' is not defined
>>> |
```

STEP 5

Return to the newly created window containing the function definitions, and click File > Save As. Name this **minimath.py** and save it in the same location as the original **basic_math.py** program. Now close the minimath.py window, so the basic_math.py window is left open.


Untitled
File Edit Format Run Options Windows Help
Save As
4letterwrd.py
Adventure.py
AlarmClock.py
Balls.py
basic_math.py
bkup.py
Booleantest.py
BoxesColours.py
CalcPi.py
Calculator.py
calendaryear.py
cards.py
Save
Cancel
Ln: 1 Col: 0

Directory: /home/pi/Documents/Python Code

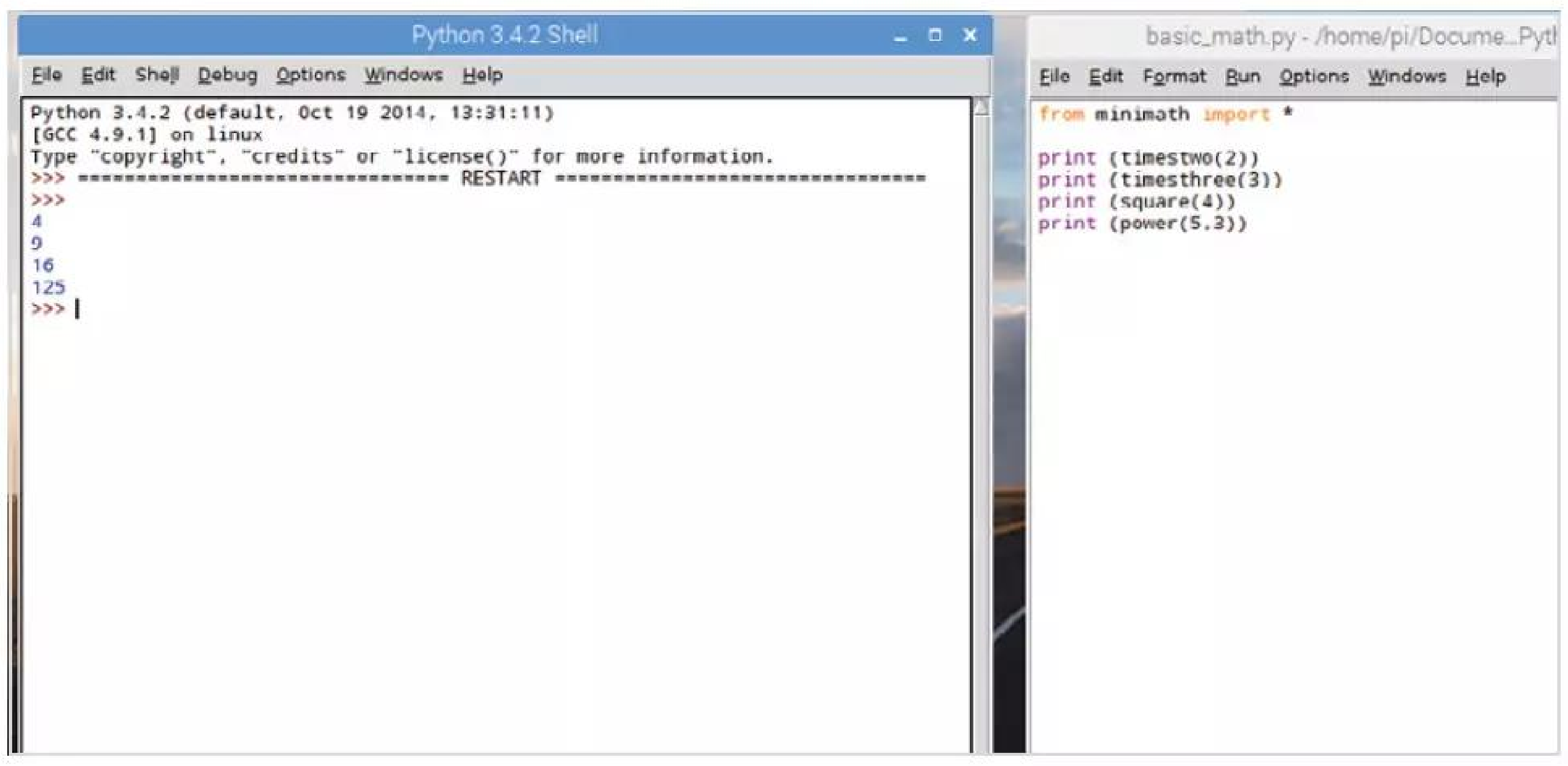
File name: minimath.py

Files of type: Python files (*.py,*.pyw)

STEP 6 Back to the basic_math.py window: at the top of the code enter:

```
from minimath import *
```

This will import the function definitions as a module. Press F5 to save and execute the program to see it in action.

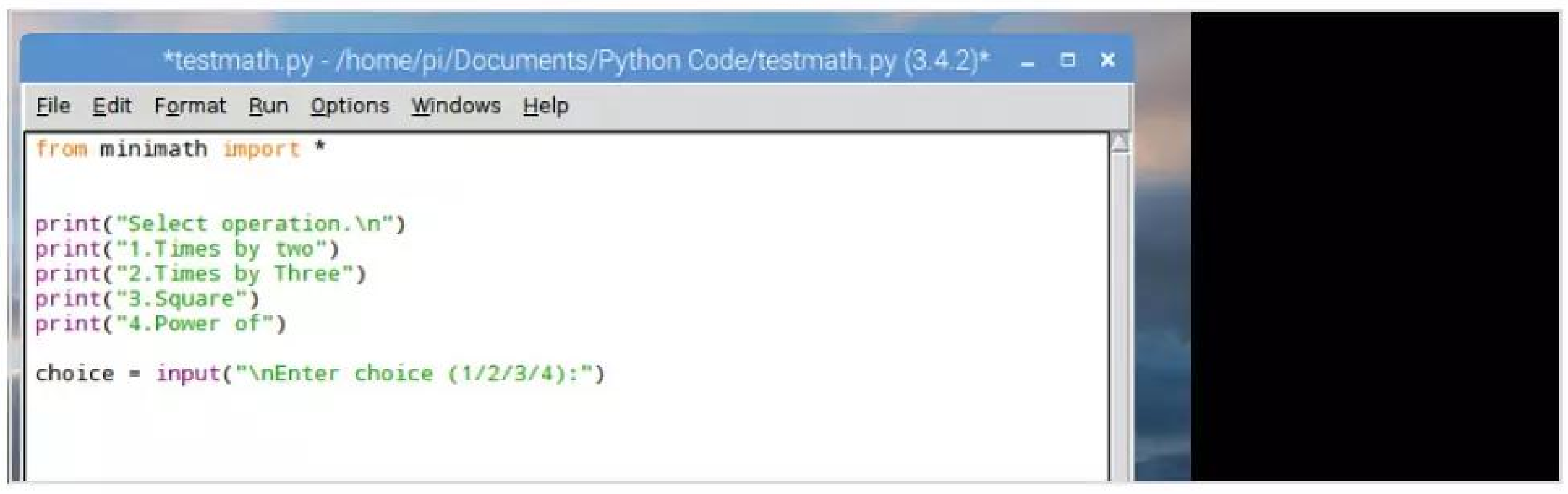

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
basic_math.py - /home/pi/Docume...Pyt
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
4
9
16
125
>>> |
```

```
from minimath import *

print (timestwo(2))
print (timesthree(3))
print (square(4))
print (power(5,3))
```

STEP 7 You can now use the code further to make the program a little more advanced, utilising the newly created module to its full. Include some user interaction. Start by creating a basic menu the user can choose from:

```
print(“Select operation.\n”)
print(“1.Times by two”)
print(“2.Times by Three”)
print(“3.Square”)
print(“4.Power of”)

choice = input(“\nEnter choice (1/2/3/4):”)
```

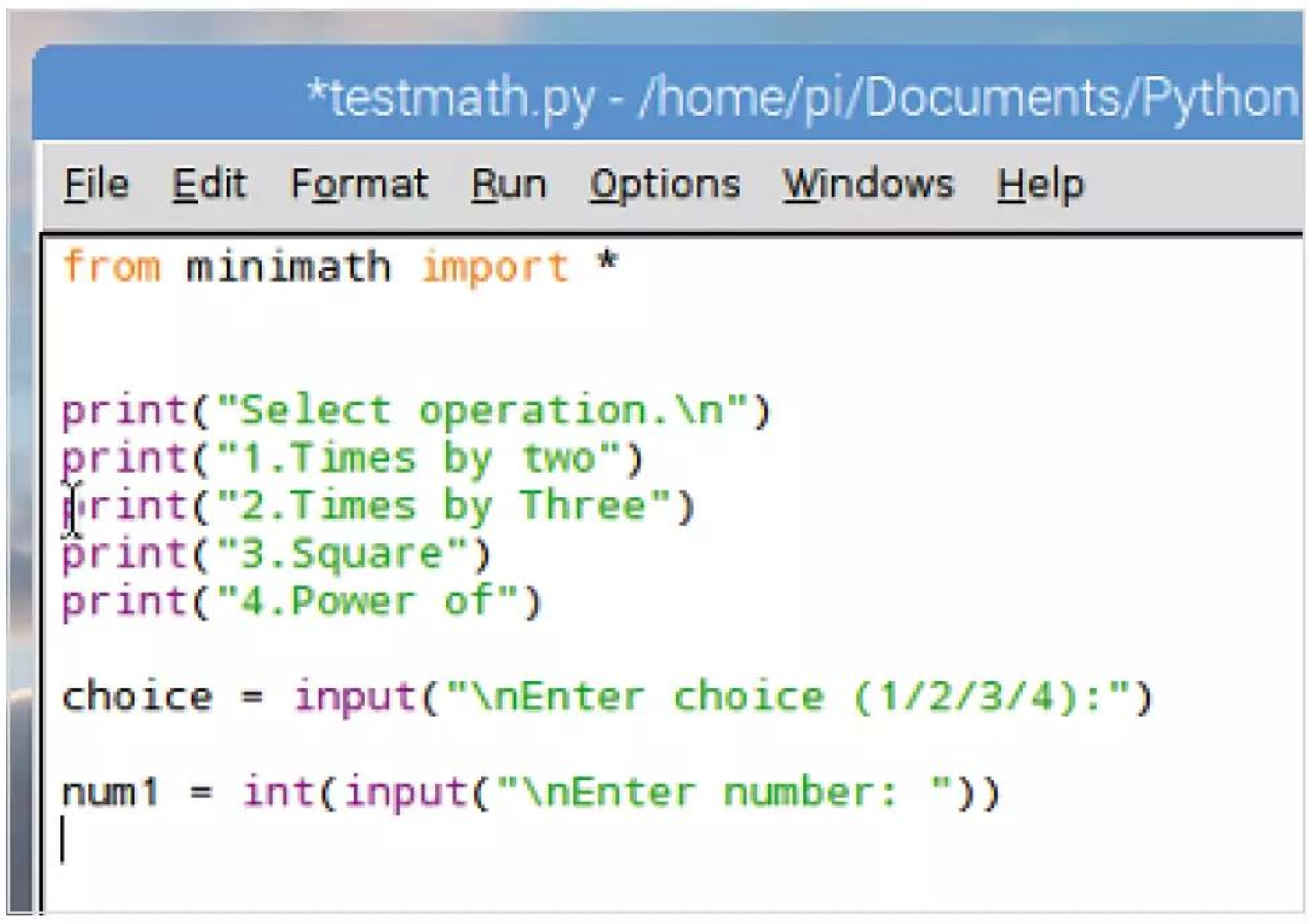

testmath.py - /home/pi/Documents/Python Code/testmath.py (3.4.2)
File Edit Format Run Options Windows Help

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")
```

STEP 8 Now we can add the user input to get the number the code will work on:

```
num1 = int(input(“\nEnter number: “))
```

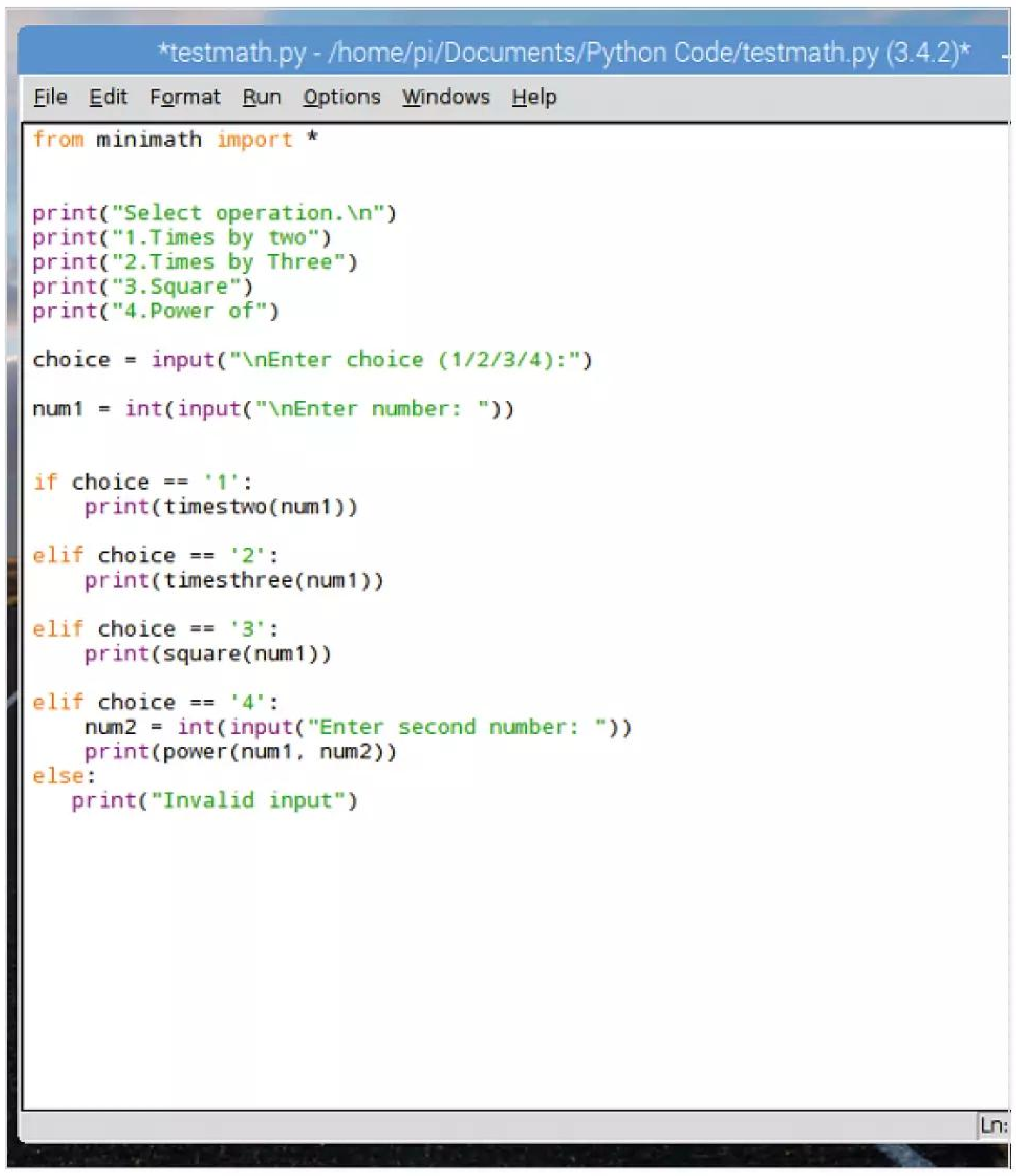
This will save the user-entered number as the variable num1.


*testmath.py - /home/pi/Documents/Python
File Edit Format Run Options Windows Help

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter number: "))
|
```

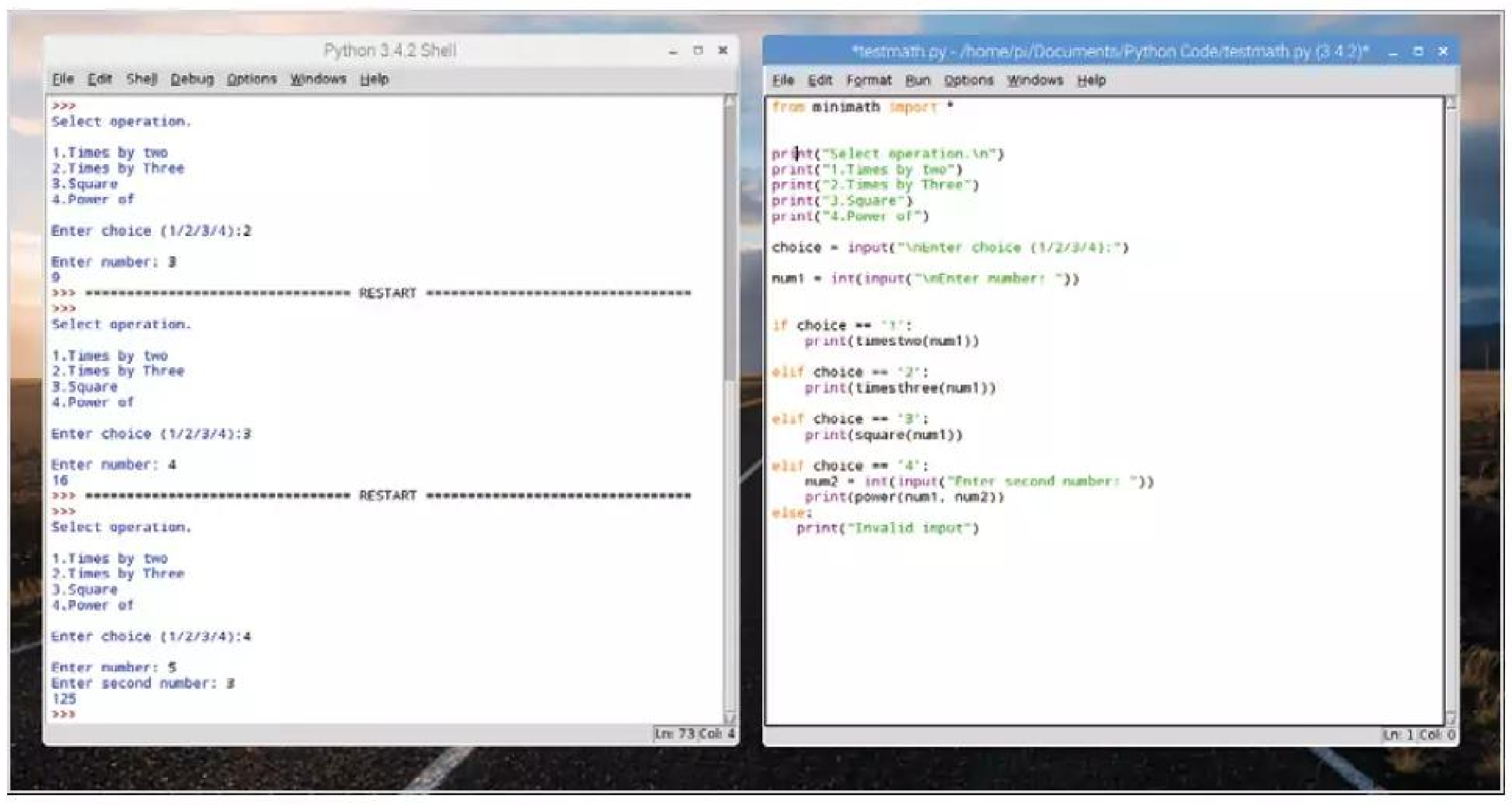
STEP 9 Finally, you can now create a range of if statements to determine what to do with the number and utilise the newly created function definitions:

```
if choice == ‘1’:
    print(timestwo(num1))

elif choice == ‘2’:
    print(timesthree(num1))

elif choice == ‘3’:
    print(square(num1))

elif choice == ‘4’:
    num2 = int(input(“Enter second number: “))
    print(power(num1, num2))
else:
   print(“Invalid input”)
```


testmath.py - /home/pi/Documents/Python Code/testmath.py (3.4.2)
File Edit Format Run Options Windows Help
Ln:

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter number: "))


if choice == '1':
    print(timestwo(num1))

elif choice == '2':
    print(timesthree(num1))

elif choice == '3':
    print(square(num1))

elif choice == '4':
    num2 = int(input("Enter second number: "))
    print(power(num1, num2))
else:
   print("Invalid input")
```

STEP 10 Note that for the last available options, the Power of choice, we’ve added a second variable, num2. This passes a second number through the function definition called power. Save and execute the program to see it in action.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 73 Col: 4
testmath.py - /home/pi/Documents/Python Code/testmath.py (3.4.2)
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
>>>
Select operation.

1.Times by two
2.Times by Three
3.Square
4.Power of

Enter choice (1/2/3/4):2

Enter number: 3
9
>>> ================================ RESTART ================================
>>>
Select operation.

1.Times by two
2.Times by Three
3.Square
4.Power of

Enter choice (1/2/3/4):3

Enter number: 4
16
>>> ================================ RESTART ================================
>>>
Select operation.

1.Times by two
2.Times by Three
3.Square
4.Power of

Enter choice (1/2/3/4):4

Enter number: 5
Enter second number: 3
125
>>>
```

```
from minimath import *


print("Select operation.\n")
print("1.Times by two")
print("2.Times by Three")
print("3.Square")
print("4.Power of")

choice = input("\nEnter choice (1/2/3/4):")

num1 = int(input("\nEnter number: "))


if choice == '1':
    print(timestwo(num1))

elif choice == '2':
    print(timesthree(num1))

elif choice == '3':
    print(square(num1))

elif choice == '4':
    num2 = int(input("Enter second number: "))
    print(power(num1, num2))
else:
   print("Invalid input")
```

C++ Input/ Output

There's a satisfying feeling when you program code that asks the user for input, then uses that input to produce something that the user can see. Even if it's simply asking for someone's name, and displaying a personal welcome message, it's a big leap forward.

User interaction, character literals, defining constants and file input and output are all covered in the following pages. All of which help you to understand how a C++ program works better.

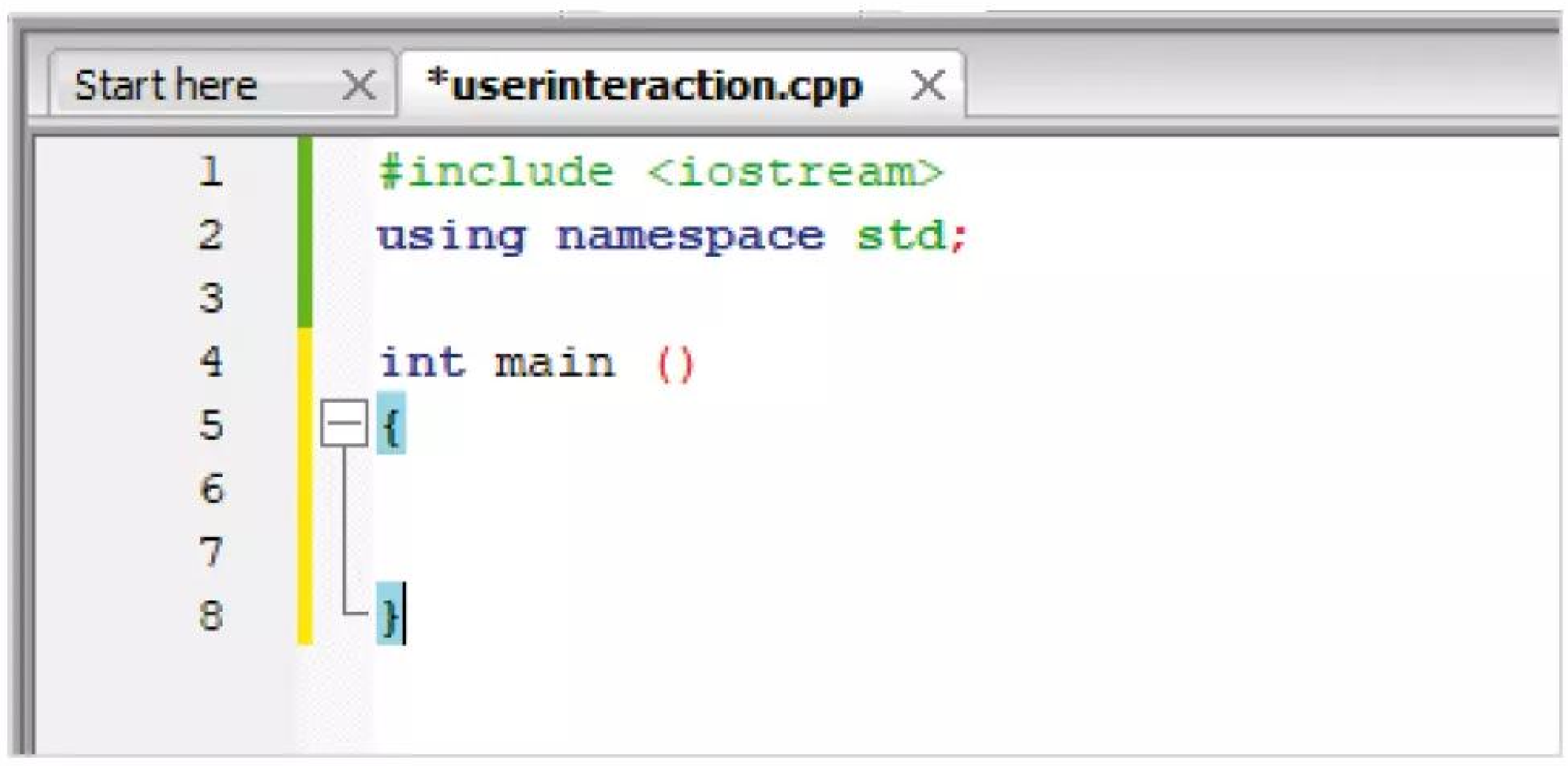
User Interaction

There's nothing quite as satisfying as creating a program that responds to you. This basic user interaction is one of the most taught aspects of any language and with it you're able to do much more than simply greet the user by name.

HELLO, DAVE

You have already used cout, the standard output stream, throughout our code. Now you're going to be using cin, the standard input stream, to prompt a user response.

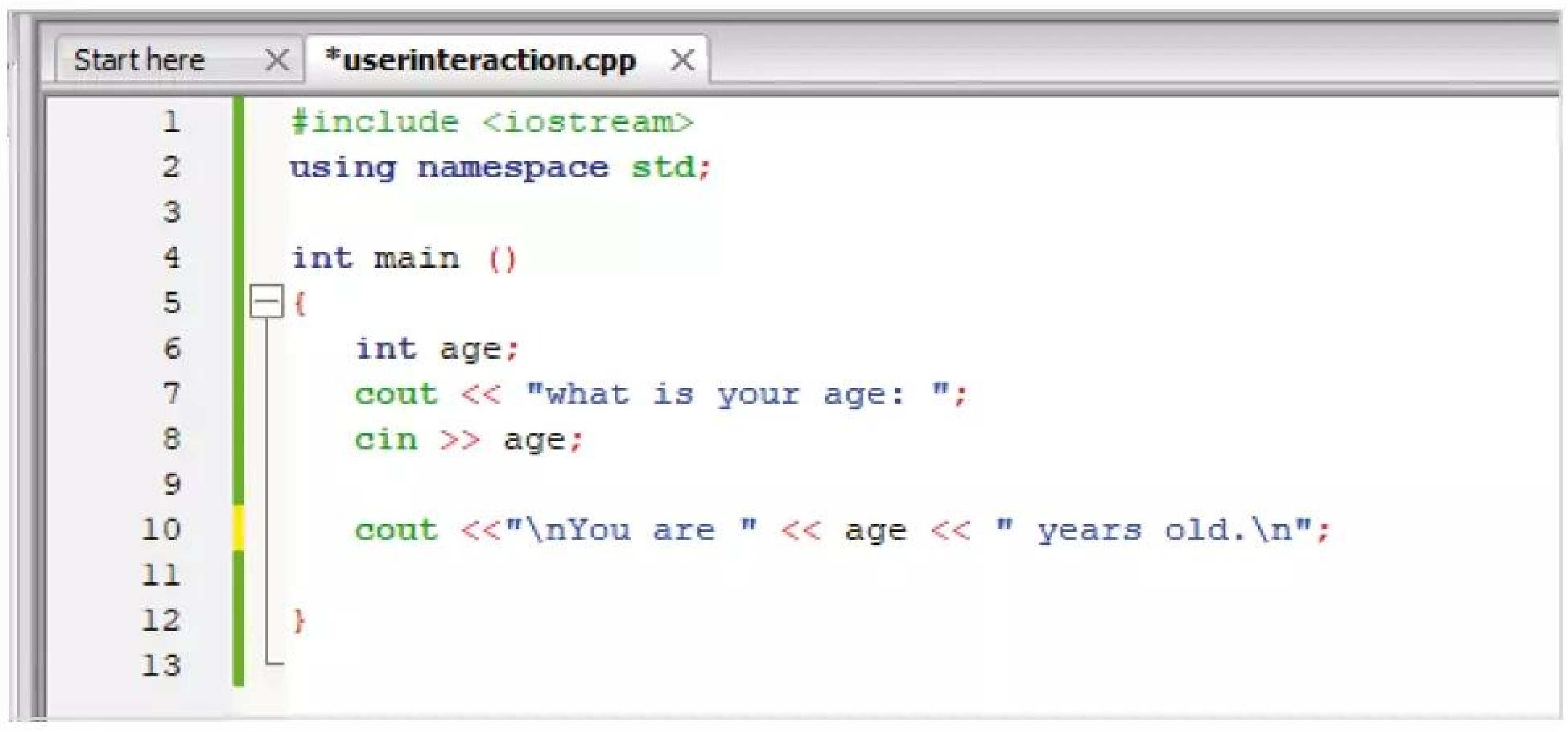
STEP 1 Anything that you want the user to input into the program needs to be stored somewhere in the system memory, so it can be retrieved and used. Therefore, any input must first be declared as a variable, so it's ready to be used by the user. Start by creating a blank C++ file with headers.



```
#include <iostream>
using namespace std;

int main ()
{


}
```

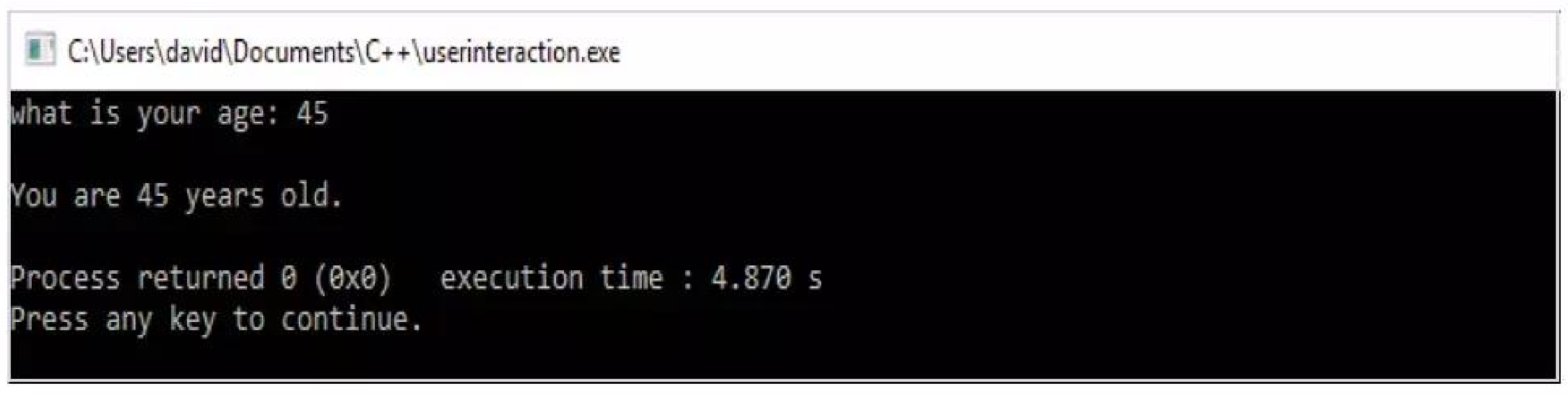
STEP 2 The data type of the variable must also match the type of input you want from the user. For example, to ask a user their age, you would use an integer like this:

```
#include <iostream>
using namespace std;

int main ()
{
   int age;
   cout << “what is your age: “;
   cin >> age;

   cout <<”\nYou are “ << age << “ years old.\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    int age;
    cout << "what is your age: ";
    cin >> age;

    cout <<"\nYou are " << age << " years old.\n";

}
```

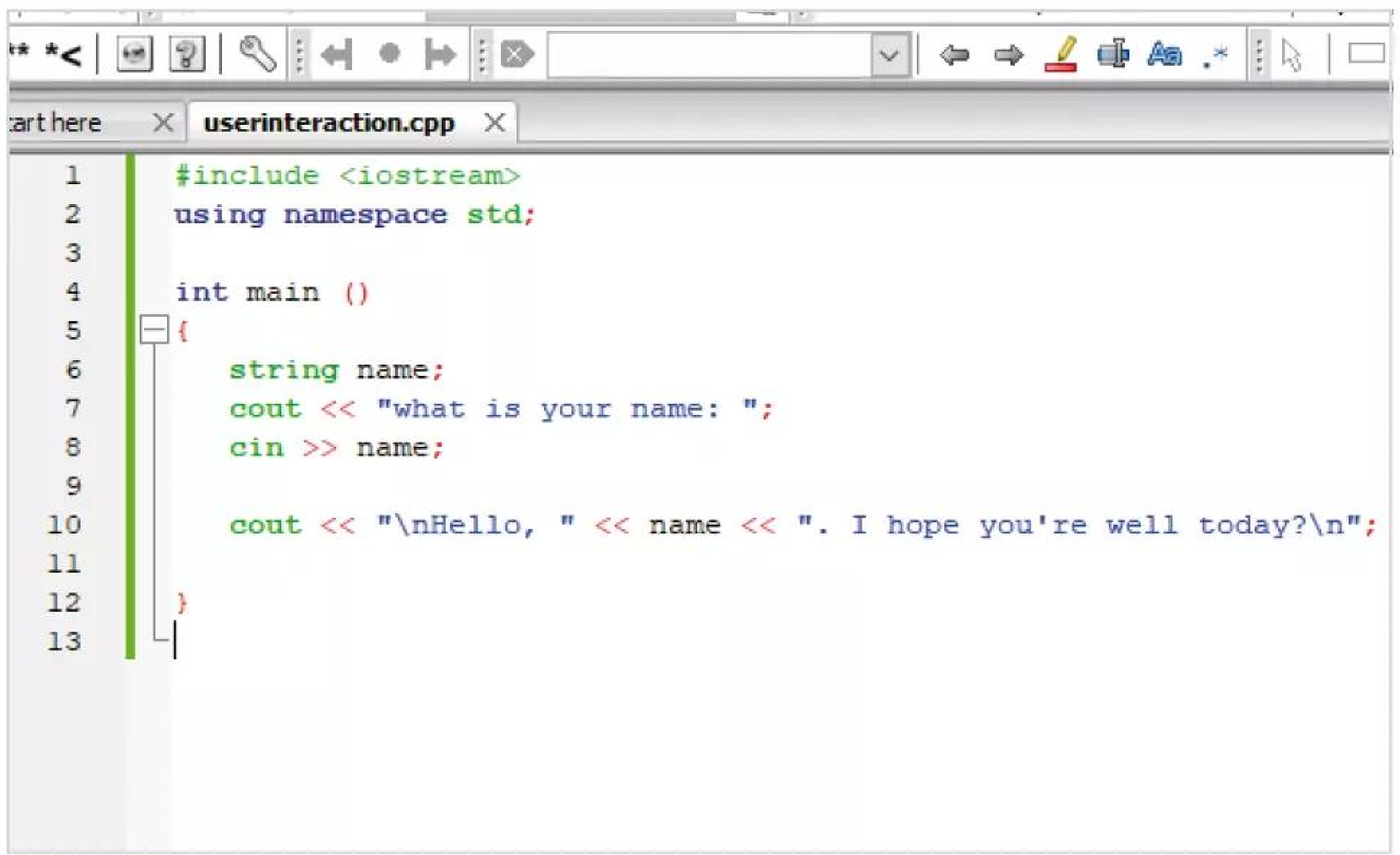
STEP 3 The cin command works in the opposite way from the cout command. With the first cout line you're outputting 'What is your age' to the screen, as indicated with the chevrons. Cin uses opposite facing chevrons, indicating an input. The input is put into the integer age and called up in the second cout command. Build and run the code.



```
what is your age: 45

You are 45 years old.

Process returned 0 (0x0)   execution time : 4.870 s
Press any key to continue.
```

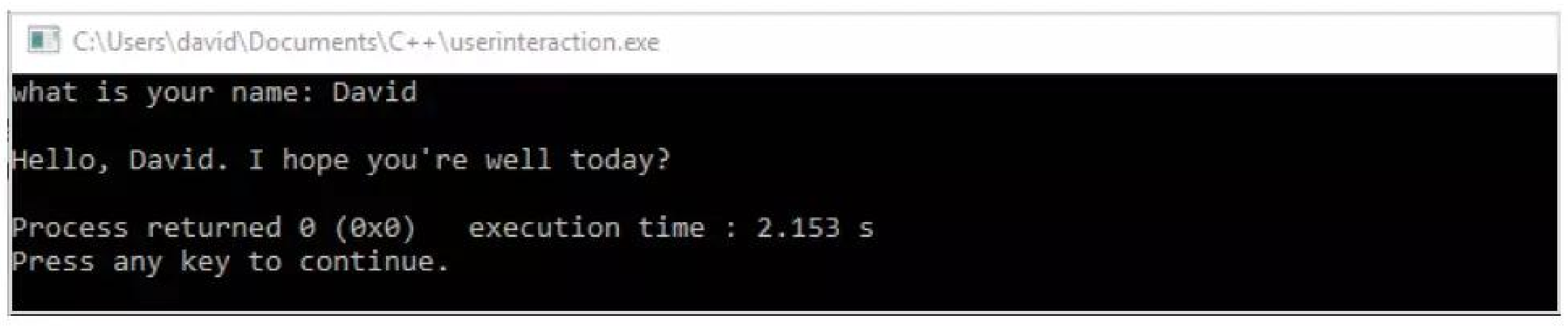
STEP 4 If you're asking a question, you need to store the input as a string; to ask the user their name, you would use:

```
#include <iostream>
using namespace std;

int main ()
{
   string name;
   cout << “what is your name: “;
   cin >> name;

   cout << “\nHello, “ << name << “. I hope you’re
well today?\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    string name;
    cout << "what is your name: ";
    cin >> name;

    cout << "\nHello, " << name << ". I hope you're well today?\n";

}
```

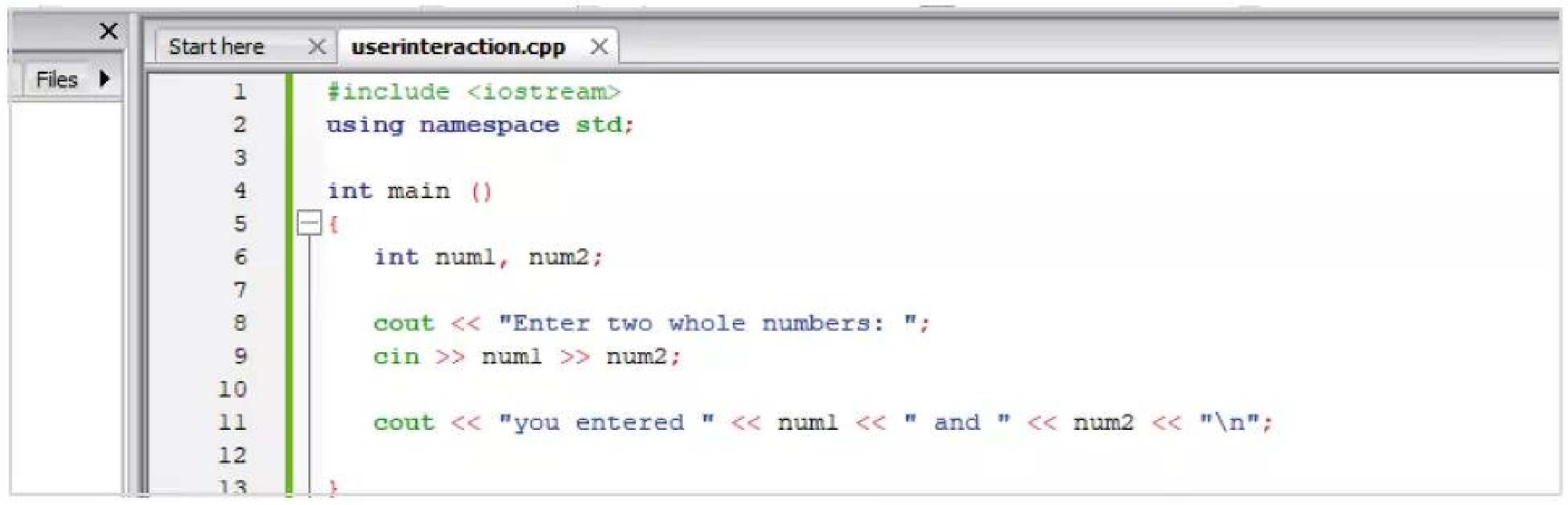
STEP 5 The principal works the same as the previous code. The user's input, their name, is stored in a string, because it contains multiple characters, and retrieved in the second cout line. As long as the variable 'name' doesn't change, then you can recall it wherever you like in your code


C:\Users\david\Documents\C++\userinteraction.exe

```
what is your name: David

Hello, David. I hope you're well today?

Process returned 0 (0x0)   execution time : 2.153 s
Press any key to continue.
```

STEP 6 You can chain input requests to the user but just make sure you have a valid variable to store the input to begin with. Let's assume you want the user to enter two whole numbers:

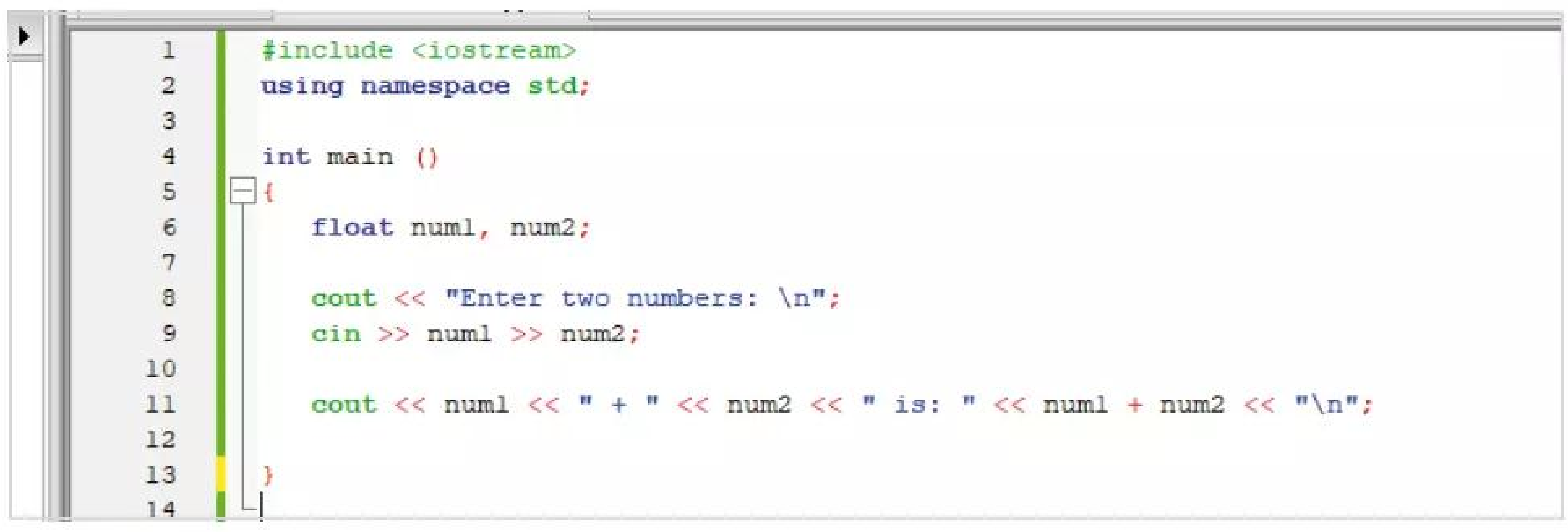
```
#include <iostream>
using namespace std;

int main ()
{
   int num1, num2;

   cout << “Enter two whole numbers: “;
   cin >> num1 >> num2;

   cout << “you entered “ << num1 << “ and “ <<
num2 << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    int num1, num2;

    cout << "Enter two whole numbers: ";
    cin >> num1 >> num2;

    cout << "you entered " << num1 << " and " << num2 << "\n";

}
```

STEP 7 Likewise, inputted data can be manipulated once you have it stored in a variable. For instance, ask the user for two numbers and do some maths on them:

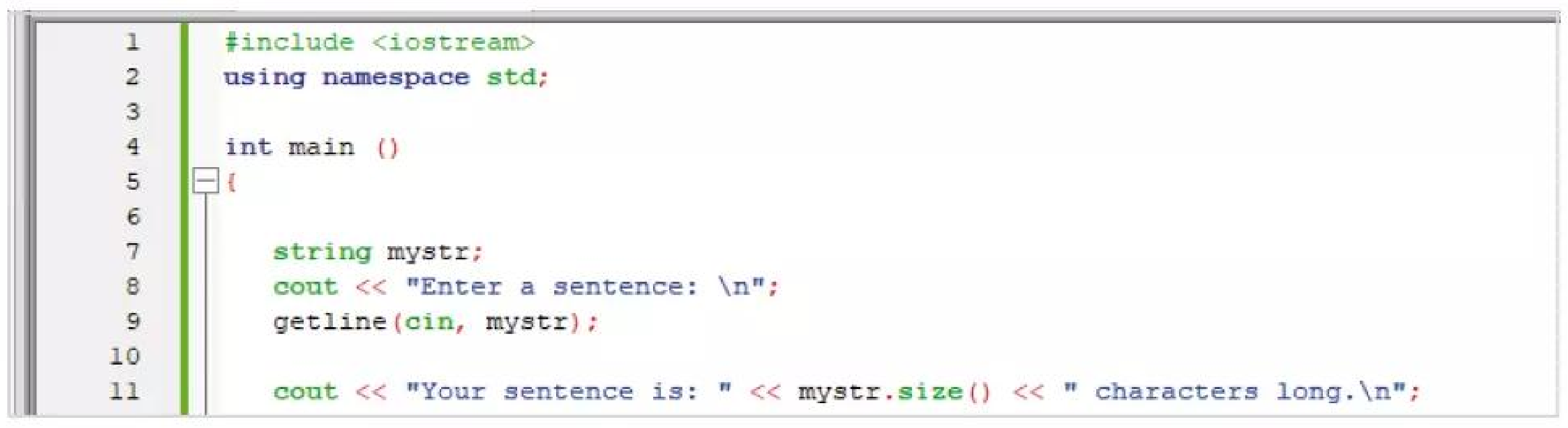
```
#include <iostream>
using namespace std;

int main ()
{
   float num1, num2;

   cout << “Enter two numbers: \n”;
   cin >> num1 >> num2;

   cout << num1 << “ + “ << num2 << “ is: “ <<
num1 + num2 << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    float num1, num2;

    cout << "Enter two numbers: \n";
    cin >> num1 >> num2;

    cout << num1 << " + " << num2 << " is: " << num1 + num2 << "\n";

}
```

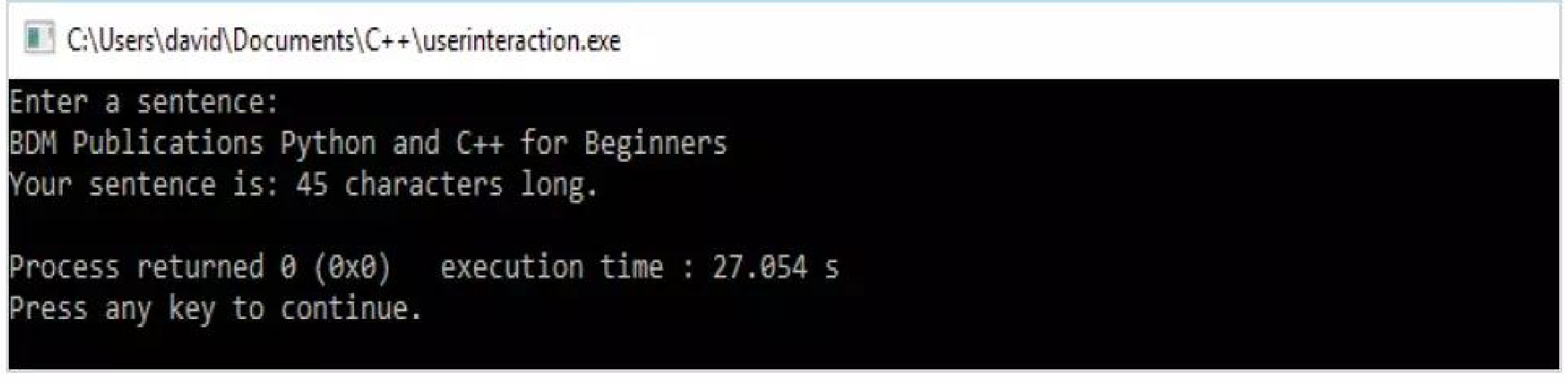
STEP 8 While cin works well for most input tasks, it does have a limitation. Cin always considers spaces as a terminator, so it's designed for just single words not multiple words. However, getline takes cin as the first argument and the variable as the second:

```
#include <iostream>
using namespace std;

int main ()
{

   string mystr;
   cout << “Enter a sentence: \n”;
   getline(cin, mystr);

   cout << “Your sentence is: “ << mystr.size() <<
“ characters long.\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{

    string mystr;
    cout << "Enter a sentence: \n";
    getline(cin, mystr);

    cout << "Your sentence is: " << mystr.size() << " characters long.\n";
```

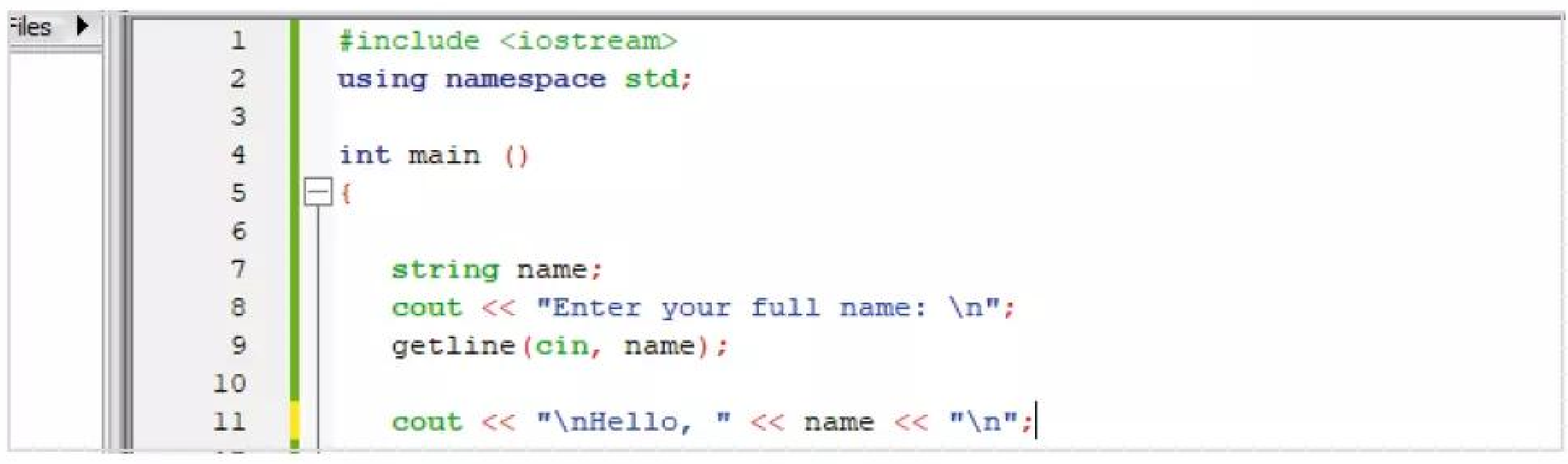
STEP 9 Build and execute the code, then enter a sentence with spaces. When you're done the code reads the number of characters. If you remove the getline line and replace it with cin >> mystr and try again, the result displays the number of characters up to the first space.


C:\Users\david\Documents\C++\userinteraction.exe

```
Enter a sentence:
BDM Publications Python and C++ for Beginners
Your sentence is: 45 characters long.

Process returned 0 (0x0)   execution time : 27.054 s
Press any key to continue.
```

STEP 10 Getline is usually a command that new C++ programmers forget to include. The terminating white space is annoying when you can't figure out why your code isn't working. In short, it's best to use getline(cin, variable) in future:

```
#include <iostream>
using namespace std;

int main ()
{

   string name;
   cout << “Enter your full name: \n”;
   getline(cin, name);

   cout << “\nHello, “ << name << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{

    string name;
    cout << "Enter your full name: \n";
    getline(cin, name);

    cout << "\nHello, " << name << "\n";
```

Character Literals

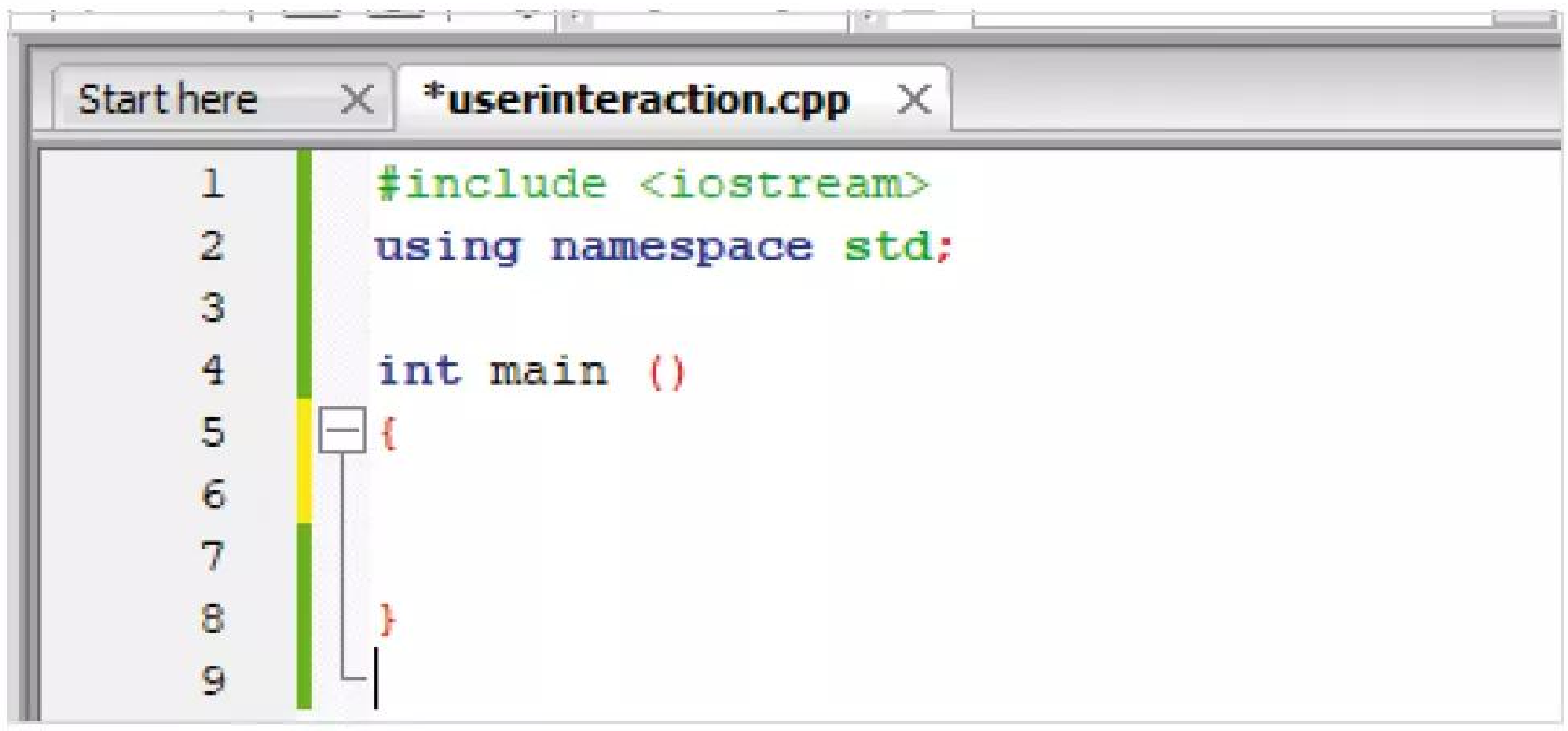
In C++ a literal is an object or variable that once defined remains the same throughout the code. However, a character literal is defined by a backslash, such as the \n you've been using at the end of a cout statement to signify a new line.

ESCAPE SEQUENCE

When used in something like a cout statement, character literals are also called Escape Sequence Codes. They allow you to insert a quote, an alert, new line and much more.

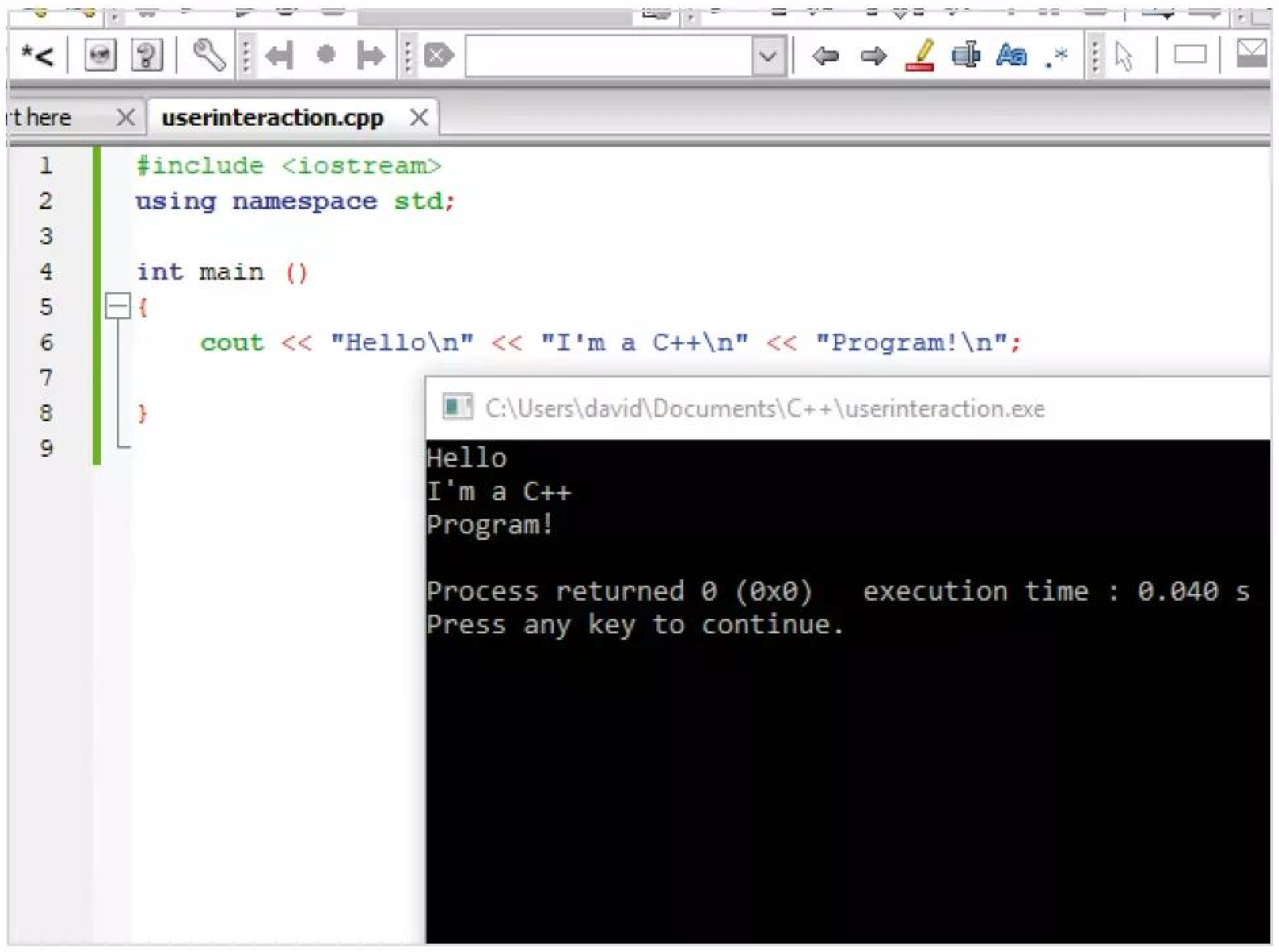
STEP 1 Create a new C++ file and enter the relevant headers:

```
#include <iostream>
using namespace std;

int main ()
{

}
```



```
#include <iostream>
using namespace std;

int main ()
{


}
```

STEP 2 You've already experienced the \n character literal placing a new line wherever it's called. The line: **cout << "Hello\n" << "I'm a C++\n" << "Program\n";** outputs three lines of text, each starting after the last \n.

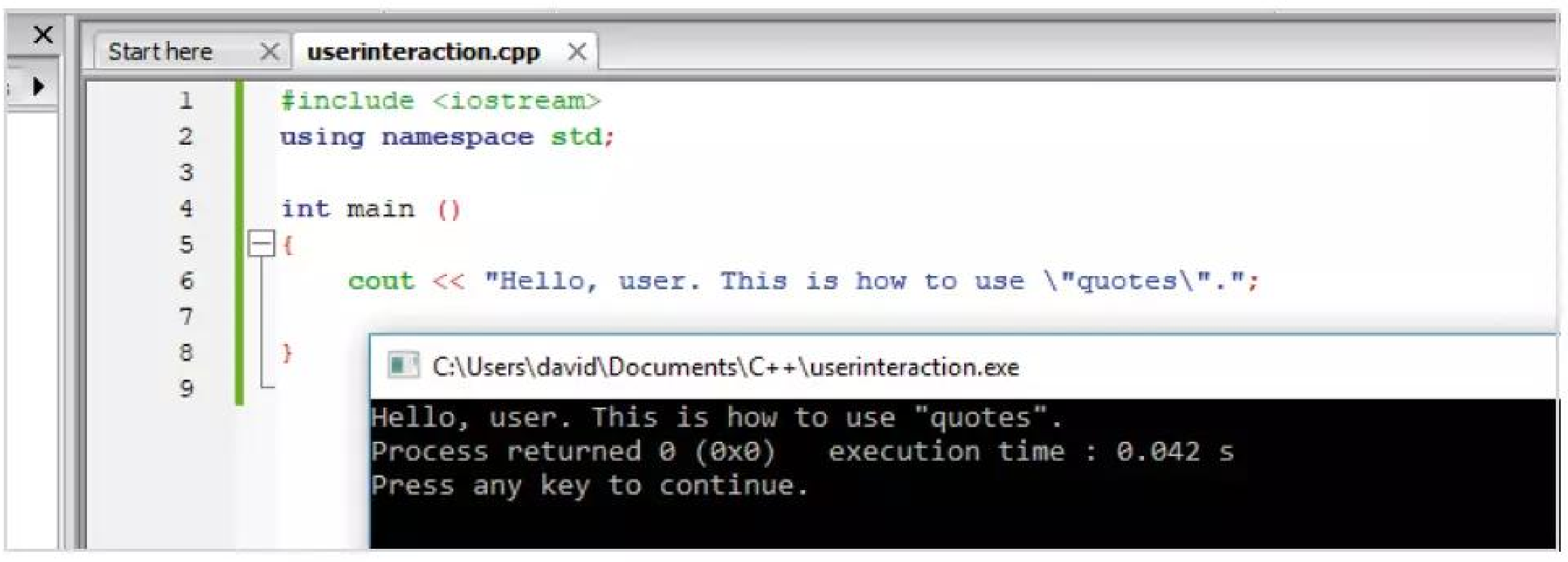


```
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello\n" << "I'm a C++\n" << "Program!\n";

}
```

```
C:\Users\david\Documents\C++\userinteraction.exe
Hello
I'm a C++
Program!

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

STEP 3 If you wanted to insert speech quotes inside a cout statement, you would have to use a backslash as it already uses quotes:

```
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello, user. This is how to use
\"quotes\".";

}
```

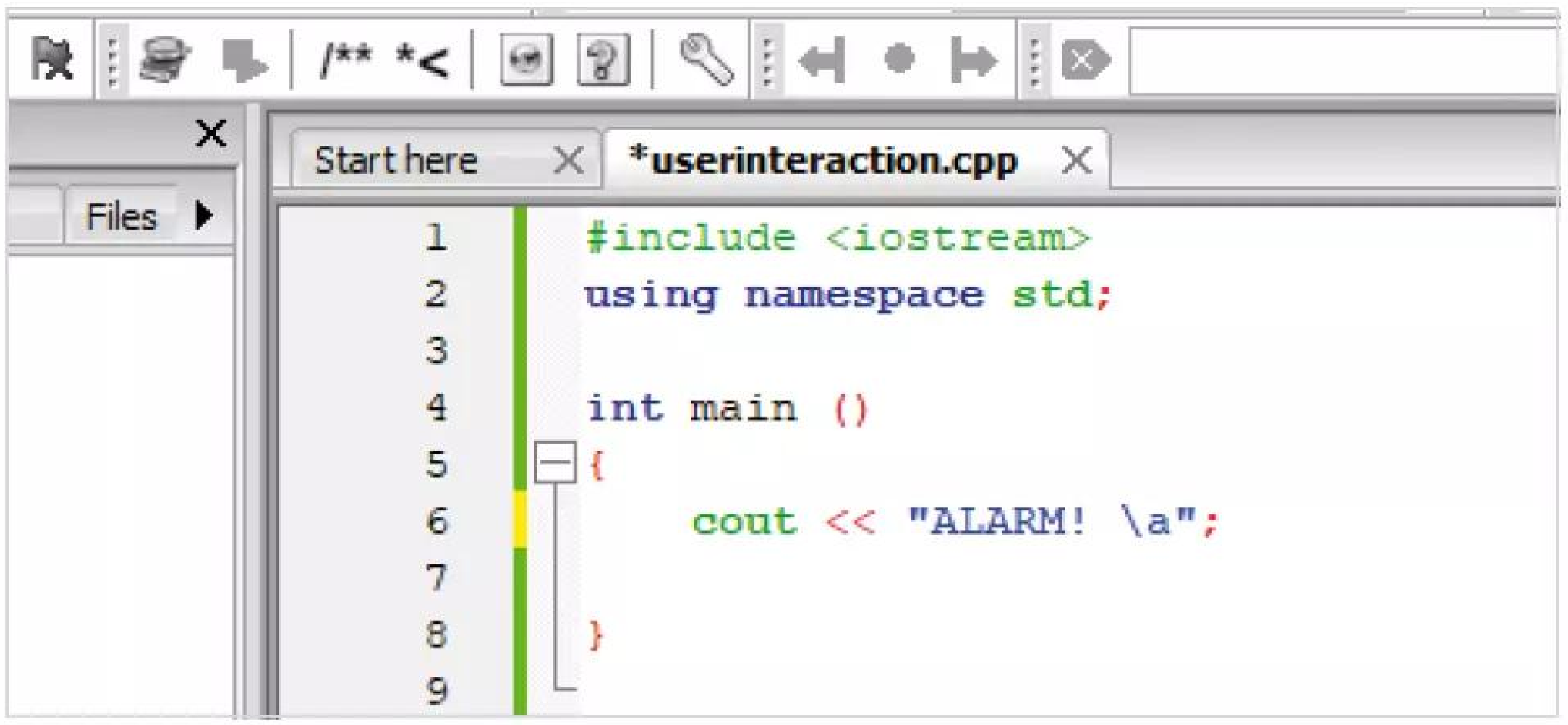


```
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello, user. This is how to use \"quotes\".";

}
```

```
C:\Users\david\Documents\C++\userinteraction.exe
Hello, user. This is how to use "quotes".
Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

STEP 4 There's even a character literal that can trigger an alarm. In Windows 10, it's the notification sound that chimes when you use \a. Try this code, and turn up your sound.

```
#include <iostream>
using namespace std;

int main ()
{
    cout << "ALARM! \a";

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    cout << "ALARM! \a";

}
```

A HANDY CHART

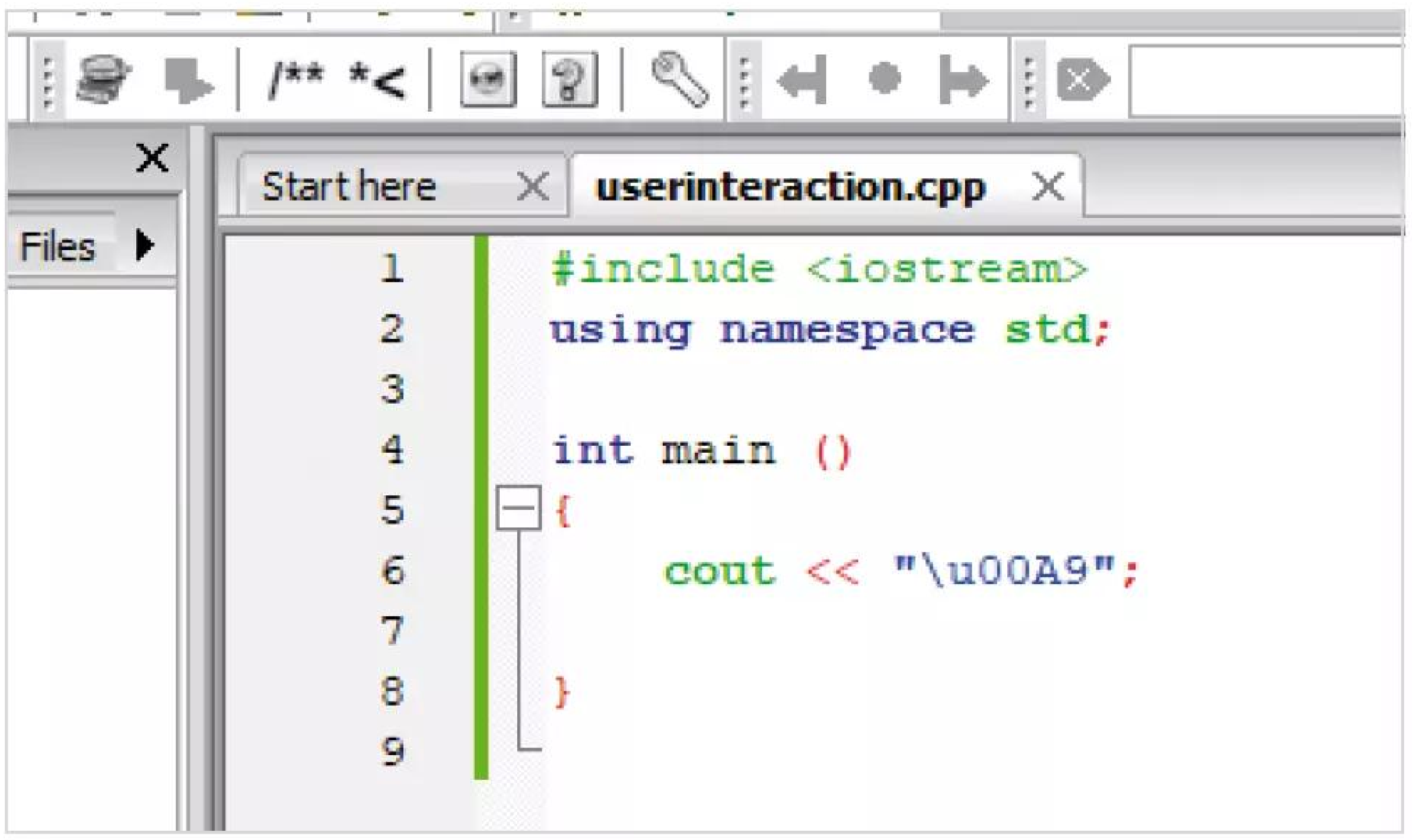
There are numerous character literals, or escape sequence codes, to choose from. We therefore thought it would be good for you to have a handy chart available, for those times when you need to insert a code.

ESCAPE SEQUENCE CODE	CHARACTER
\\	Backslash
\'	Single Quote
\"	Double Quote (Speech Marks)
\?	Question Mark
\a	Alert/Alarm
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab
\0	Null Character
\uxxxx	Unicode (UTF-8)
\Uxxxxxxxx	Unicode (UTF-16)

UNICODE CHARACTERS (UTF-8)

Unicode characters are symbols or characters that are standard across all platforms. For example, the copyright symbol, that can be entered via the keyboard by entering the Unicode code, followed by ALT+X. In the case of the copyright symbol enter: 00A9 Alt+X. In C++ code, you would enter:

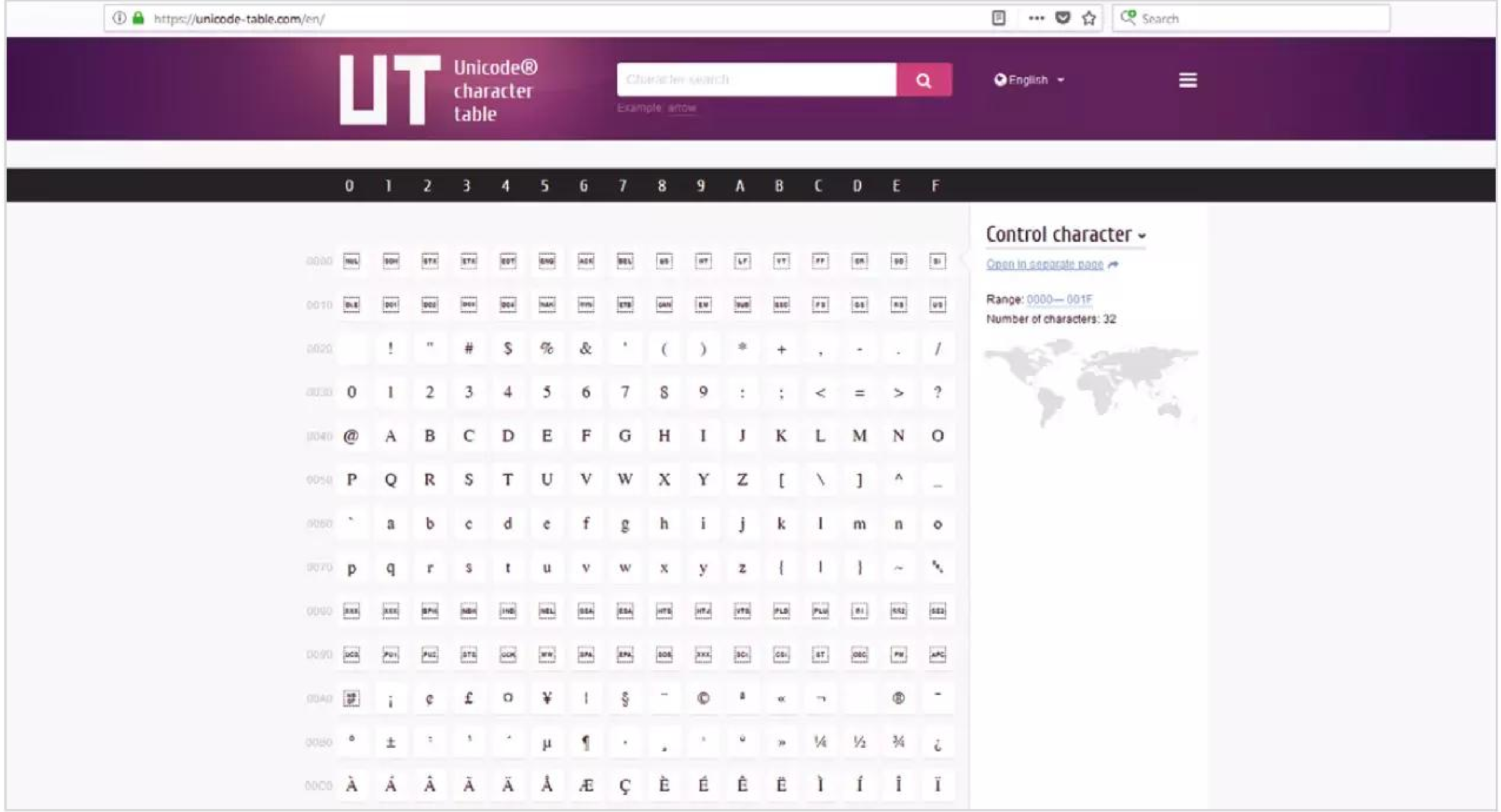
```
#include <iostream>
using namespace std;

int main ()
{
    cout << “\u00A9”;
```


Start here
userinteraction.cpp
Files

```
#include <iostream>
using namespace std;

int main ()
{
    cout << "\u00A9";

}

```

UNICODE CHARACTER TABLE

A complete list of the available Unicode characters can be found at **www.unicode-table.com/en/**. Hover your mouse over the character to see its unique code to enter in C++.


https://unicode-table.com/en/
Unicode® character table
English
Control character
Open in separate page

Range: 0000— 001F
Number of characters: 32

Defining Constants

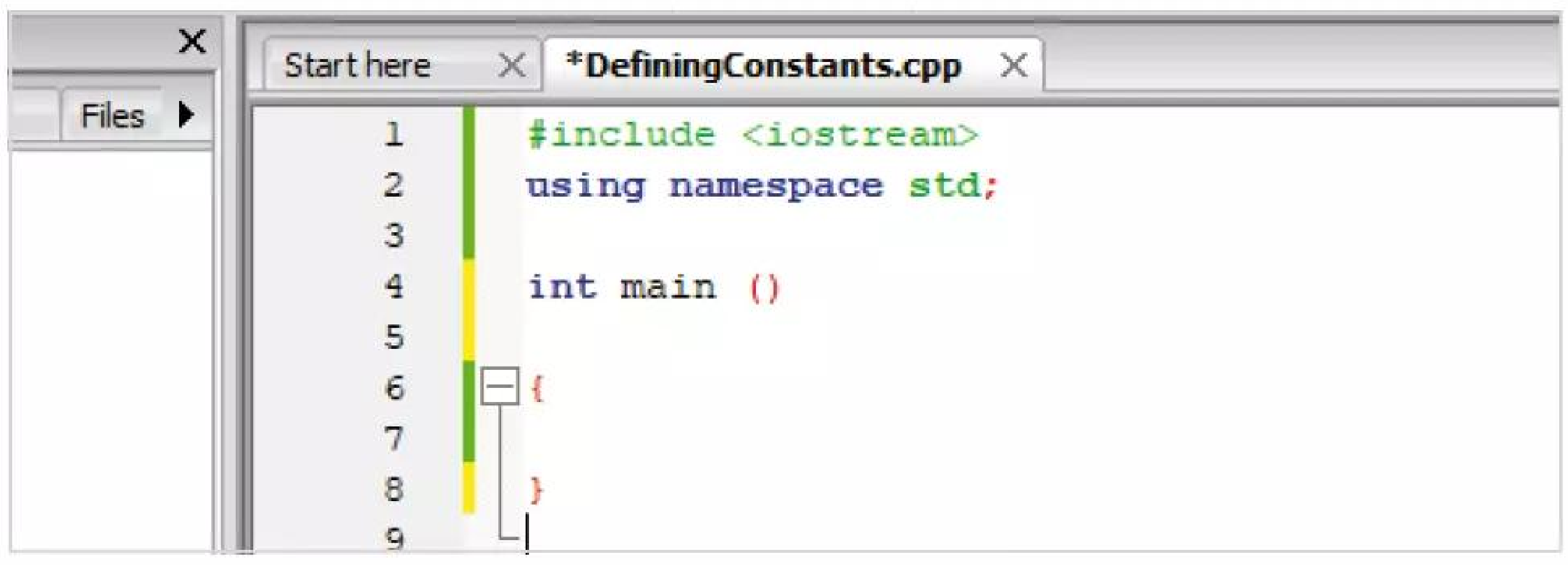
Constants are fixed values in your code. They can be any basic data type but as the name suggests their value remains constant throughout the entire code. There are two separate ways to define a constant in C++, the #define pre-processor and const.

#DEFINE

The pre-processors are instructions to the compiler to pre-process the information before it goes ahead and compiles the code. #include is a pre-processor as is #define.

STEP 1 You can use the #define pre-processor to define any constants you want in our code. Start by creating a new C++ file complete with the usual headers:

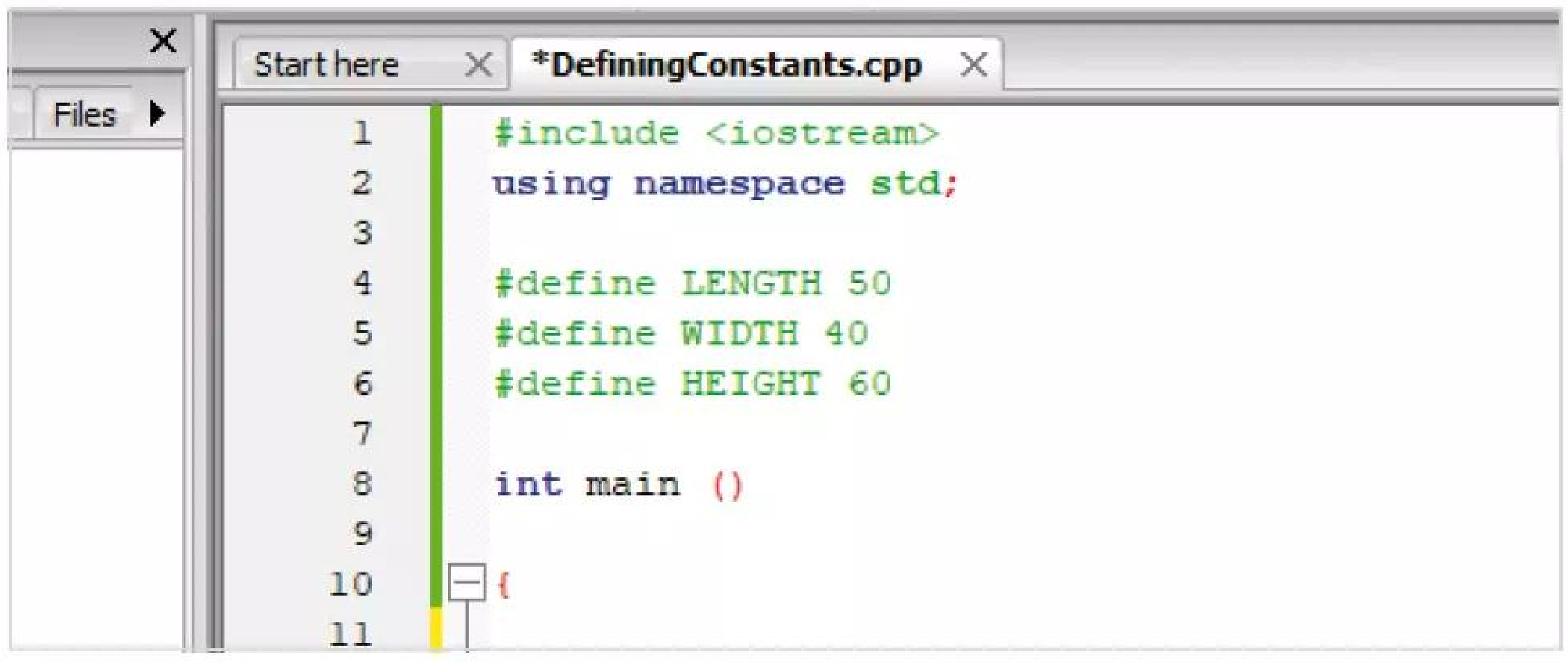
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{

}
```

STEP 2 Now let's assume your code has three different constants: length, width and height. You can define them with:

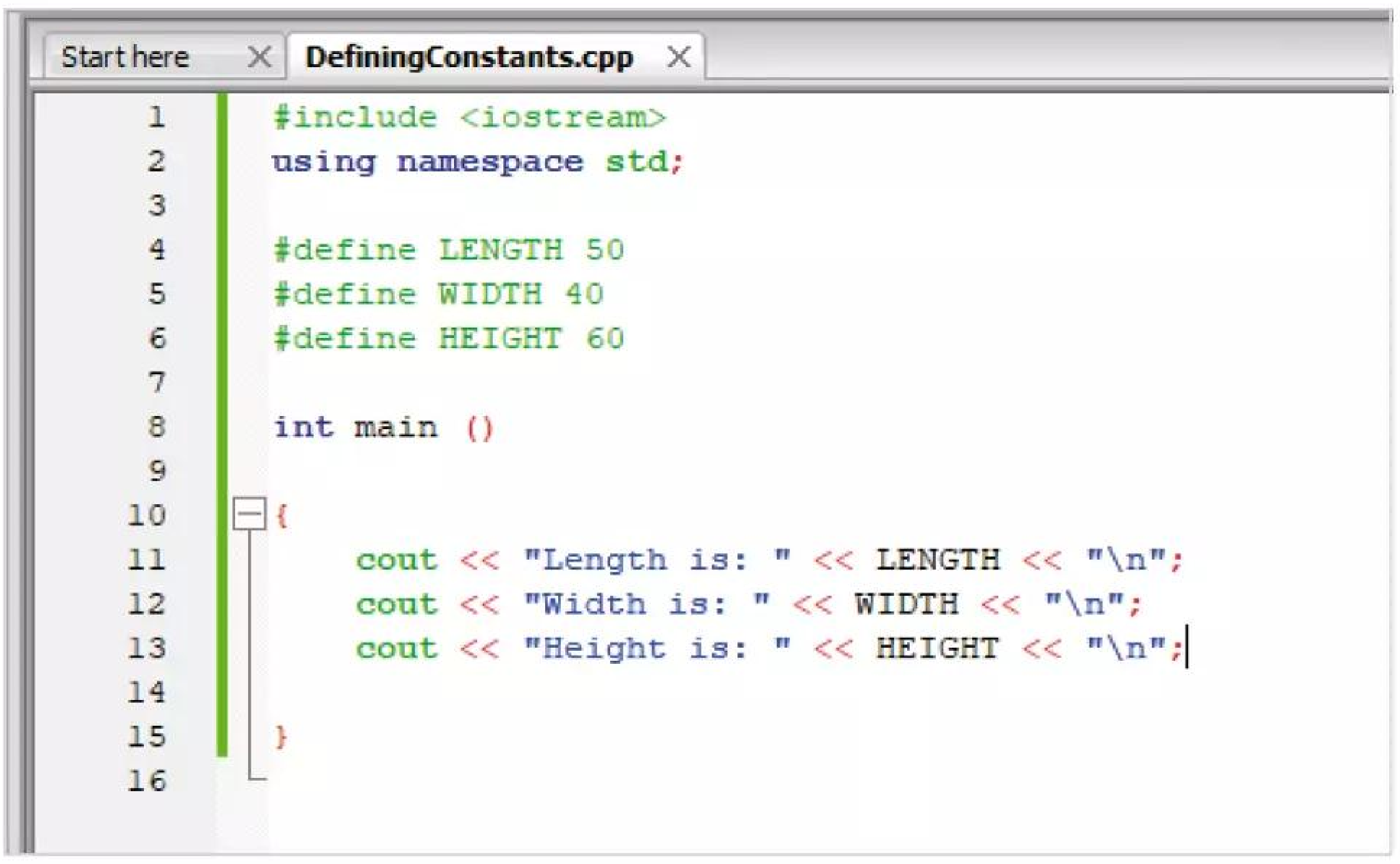
```
#include <iostream>
using namespace std;
#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{

}
```



```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{
```

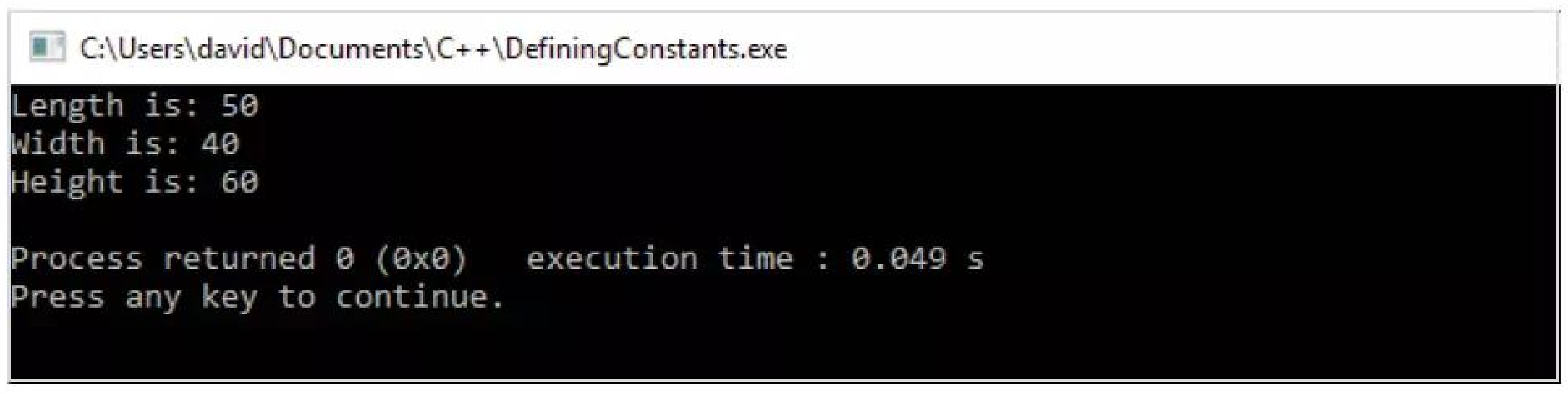
STEP 3 Note the capitals for defined constants, it's considered good programming practise to define all constants in capitals. Here, the assigned values are 50, 40 and 60, so let's call them up:

```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{
    cout << “Length is: “ << LENGTH << “\n”;
    cout << “Width is: “ << WIDTH << “\n”;
    cout << “Height is: “ << HEIGHT << “\n”;
}
```



```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60

int main ()

{
    cout << "Length is: " << LENGTH << "\n";
    cout << "Width is: " << WIDTH << "\n";
    cout << "Height is: " << HEIGHT << "\n";

}
```

STEP 4 Build and run the code. Just as expected, it displays the values for each of the constants created. It's worth noting that you don't need a semicolon when you're defining a constant with the #define keyword.

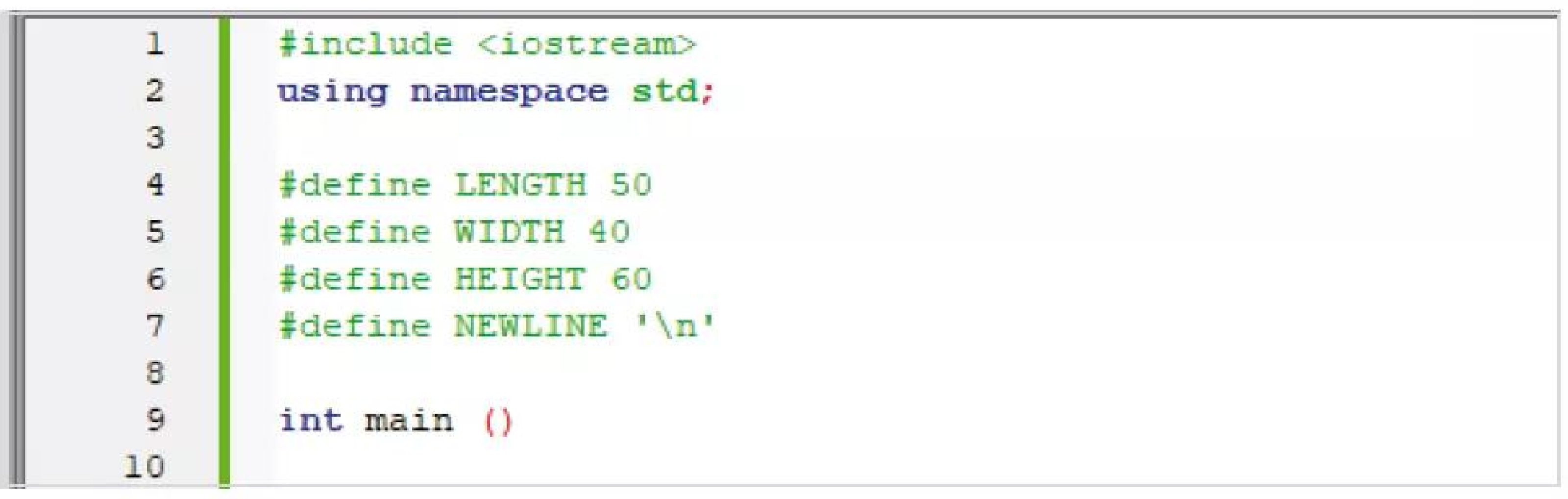


```
Length is: 50
Width is: 40
Height is: 60

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

STEP 5

You can also define other elements as a constant. For example, instead of using \n for a newline in the cout statement, you can define it at the start of the code:

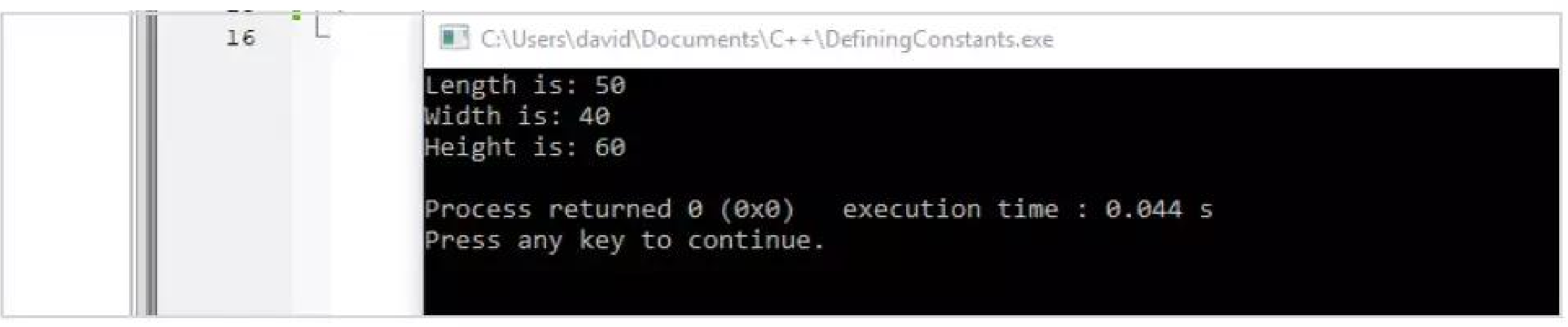
```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60
#define NEWLINE ‘\n’

int main ()

{
    cout << “Length is: “ << LENGTH << NEWLINE;
    cout << “Width is: “ << WIDTH << NEWLINE;
    cout << “Height is: “ << HEIGHT << NEWLINE;
}
```



```
#include <iostream>
using namespace std;

#define LENGTH 50
#define WIDTH 40
#define HEIGHT 60
#define NEWLINE '\n'

int main ()

```

STEP 6

The code, when built and executed, does exactly the same as before, using the new constant NEWLINE to insert a newline in the cout statement. Incidentally, creating a newline constant isn’t a good idea unless you’re making it smaller than \n or even the endl command.

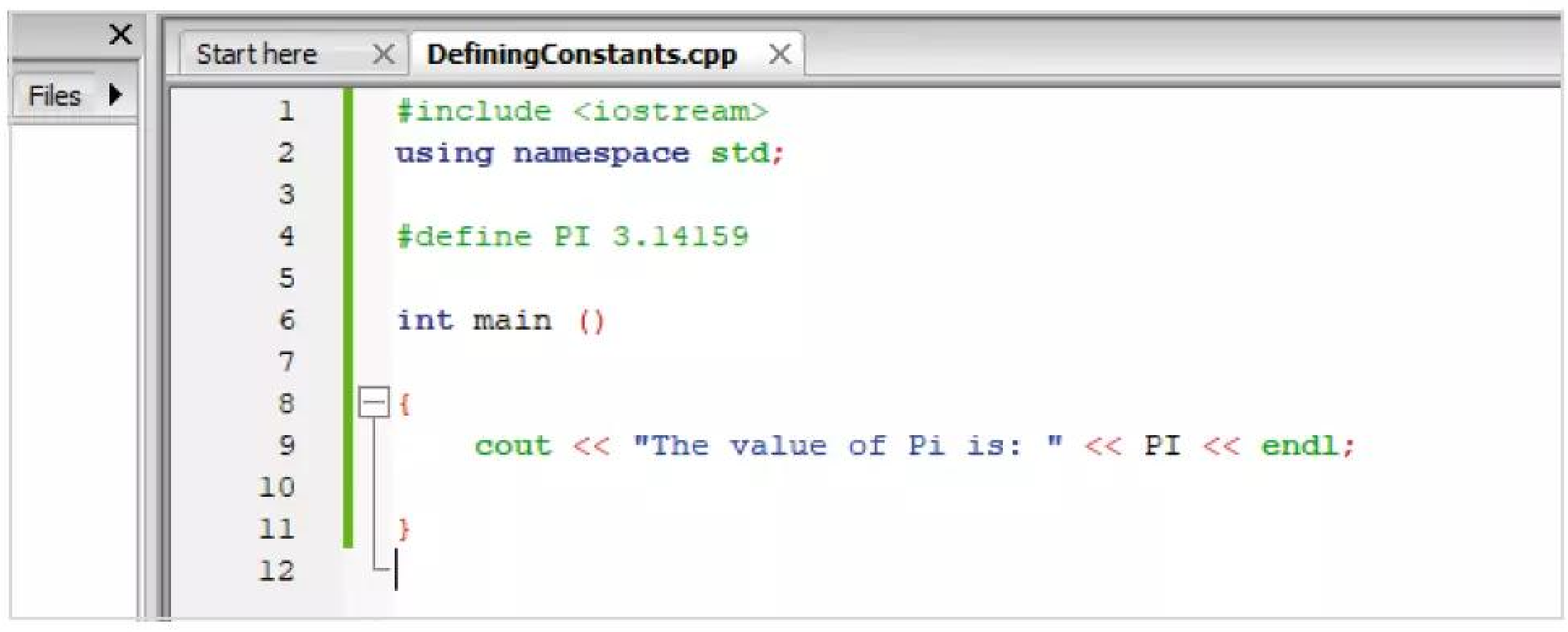


```
Length is: 50
Width is: 40
Height is: 60

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

STEP 7

Defining a constant is a good way of initialising your base values at the start of your code. You can define that your game has three lives, or even the value of PI without having to call up the C++ math library:

```
#include <iostream>
using namespace std;

#define PI 3.14159

int main ()

{
    cout << “The value of Pi is: “ << PI << endl;

}
```

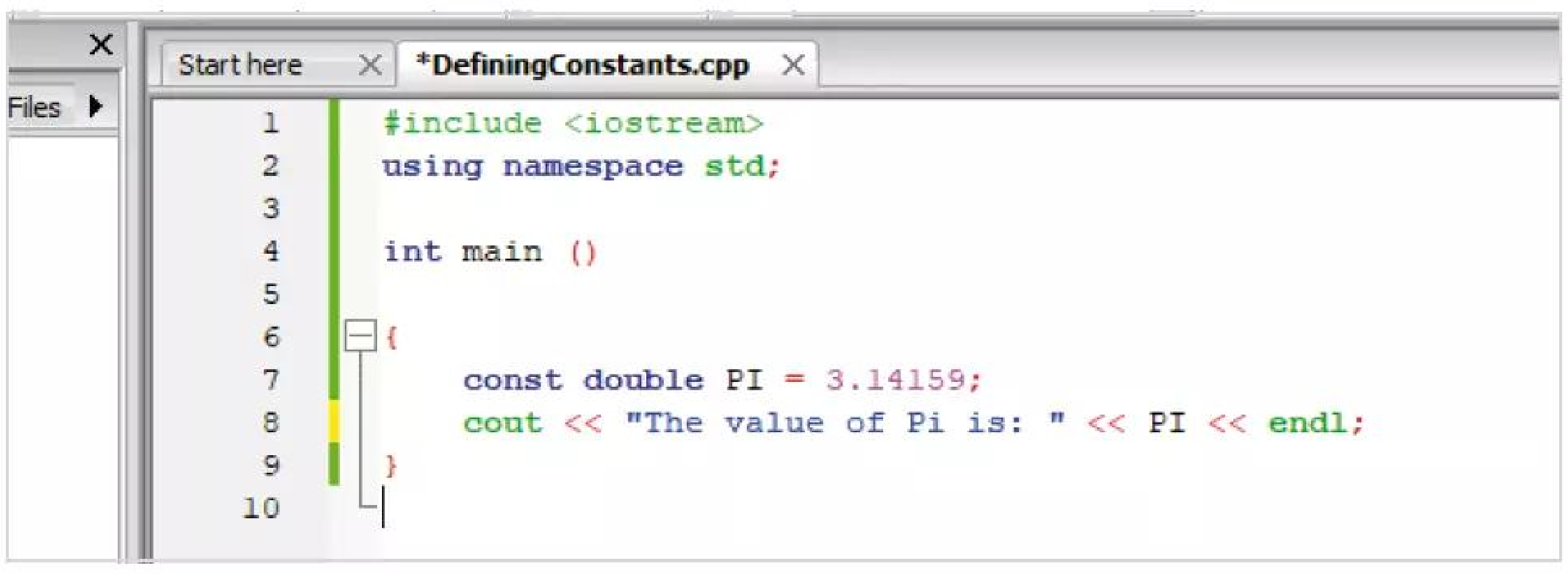


```
#include <iostream>
using namespace std;

#define PI 3.14159

int main ()

{
    cout << "The value of Pi is: " << PI << endl;

}
```

STEP 8

Another method of defining a constant is with the const keyword. Use const together with a data type, variable and value: const type variable = value. Using Pi as an example:

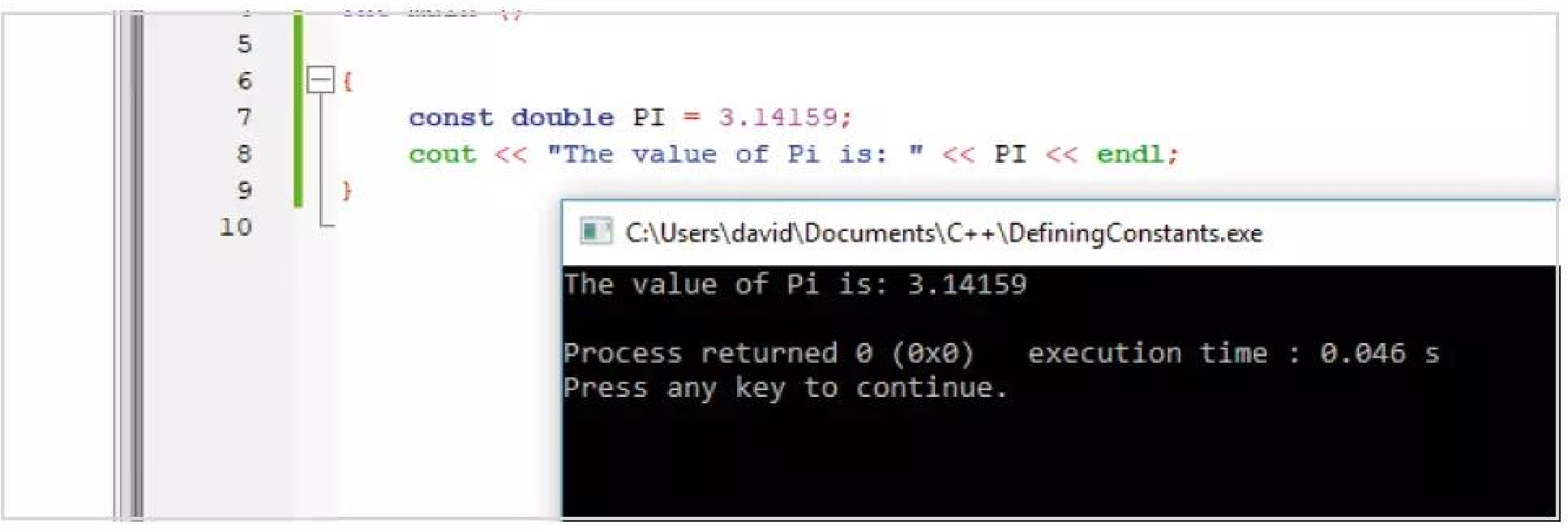
```
#include <iostream>
using namespace std;

int main ()

{
    const double PI = 3.14159;
    cout << “The value of Pi is: “ << PI << endl;
}
```



```
#include <iostream>
using namespace std;

int main ()

{
    const double PI = 3.14159;
    cout << "The value of Pi is: " << PI << endl;
}
```

STEP 9

Because you’re using const within the main block of code, you need to finish the line with a semicolon. You can use either, as long as the names and values don’t clash, but it’s worth mentioning that #define requires no memory, so if you’re coding to a set amount of memory, #define is your best bet.



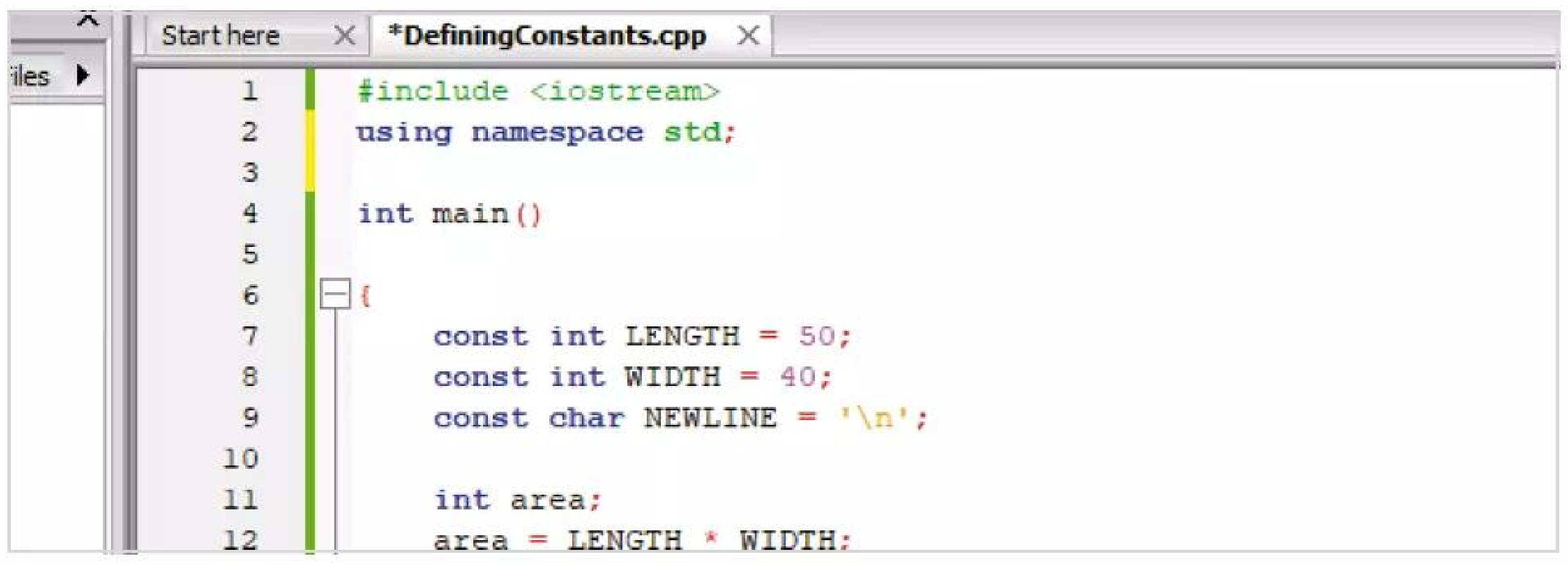
```
{
    const double PI = 3.14159;
    cout << "The value of Pi is: " << PI << endl;
}
```

```
The value of Pi is: 3.14159

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

STEP 10

Const works in much the same way as #define. You can create static integers and even newlines:

```
#include <iostream>
using namespace std;

int main()

{
    const int LENGTH = 50;
    const int WIDTH = 40;
    const char NEWLINE = ‘\n’;

    int area;
    area = LENGTH * WIDTH;

    cout << “Area is: “ << area << NEWLINE;

}
```



```
#include <iostream>
using namespace std;

int main()

{
    const int LENGTH = 50;
    const int WIDTH = 40;
    const char NEWLINE = '\n';

    int area;
    area = LENGTH * WIDTH;
```

File Input/Output

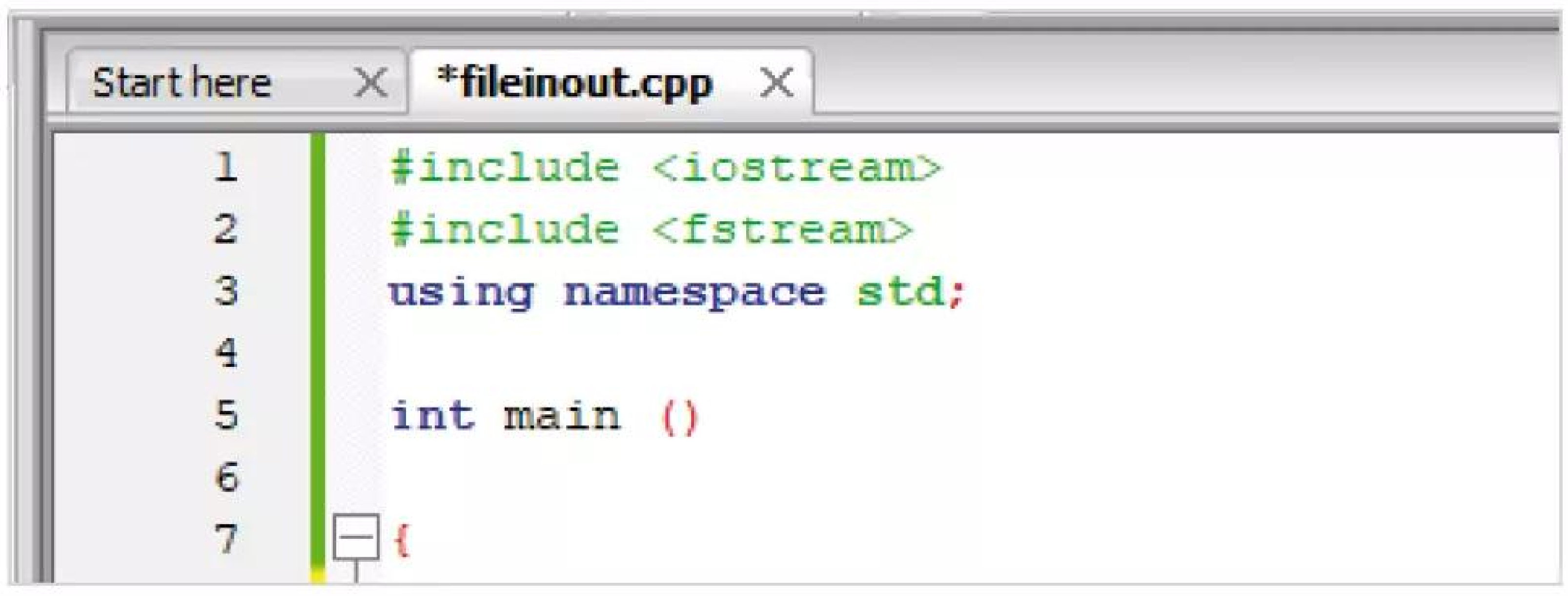
The standard iostream library provides C++ coders with the cin and cout input and output functionality. However, to be able to read and write from a file you need to utilise another C++ library, called fstream.

FSTREAMS

There are two main data types within the fstream library that are used to open a file, read from it and write to it, ofstream and ifstream. Here's how they work.

STEP 1 The first task is to create a new C++ file and along with the usual headers you need to include the new fstream header:

```
#include <iostream>
#include <fstream>
Using namespace std;

int main ()

{


}
```

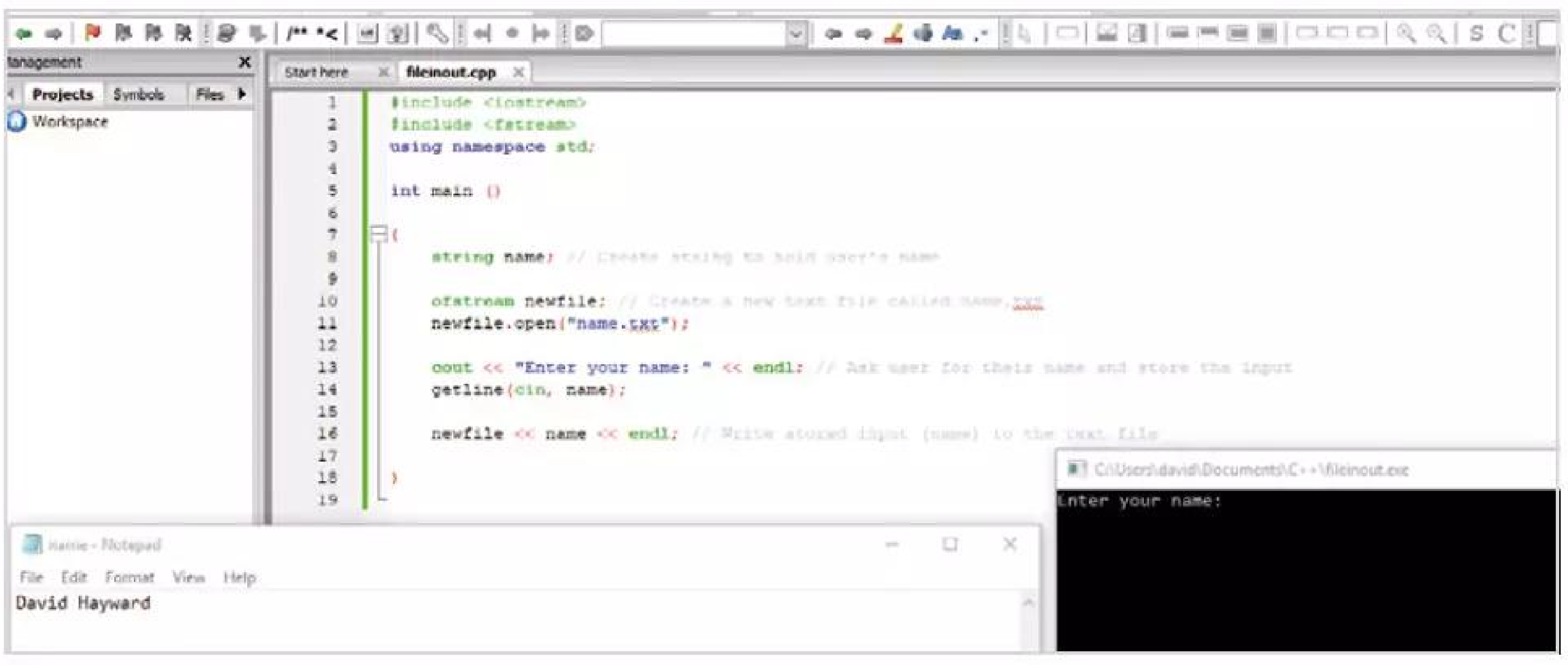


```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{
```

STEP 2 Begin by asking a user for their name and writing that information to a file. You need the usual string to store the name, and getline to accept the input from the user.

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string name;

    ofstream newfile;
    newfile.open("name.txt");

    cout << "Enter your name: " << endl;
    getline(cin, name);

    newfile << name << endl;

    newfile.close();

}
```

STEP 3 We've included comments in the screenshot of step 2 to help you understand the process. You created a string called name, to store the user's inputted name. You also created a text file called name.txt (with the ofstream newfile and newfile.open lines), asked the user for their name and stored it and then written the data to the file.



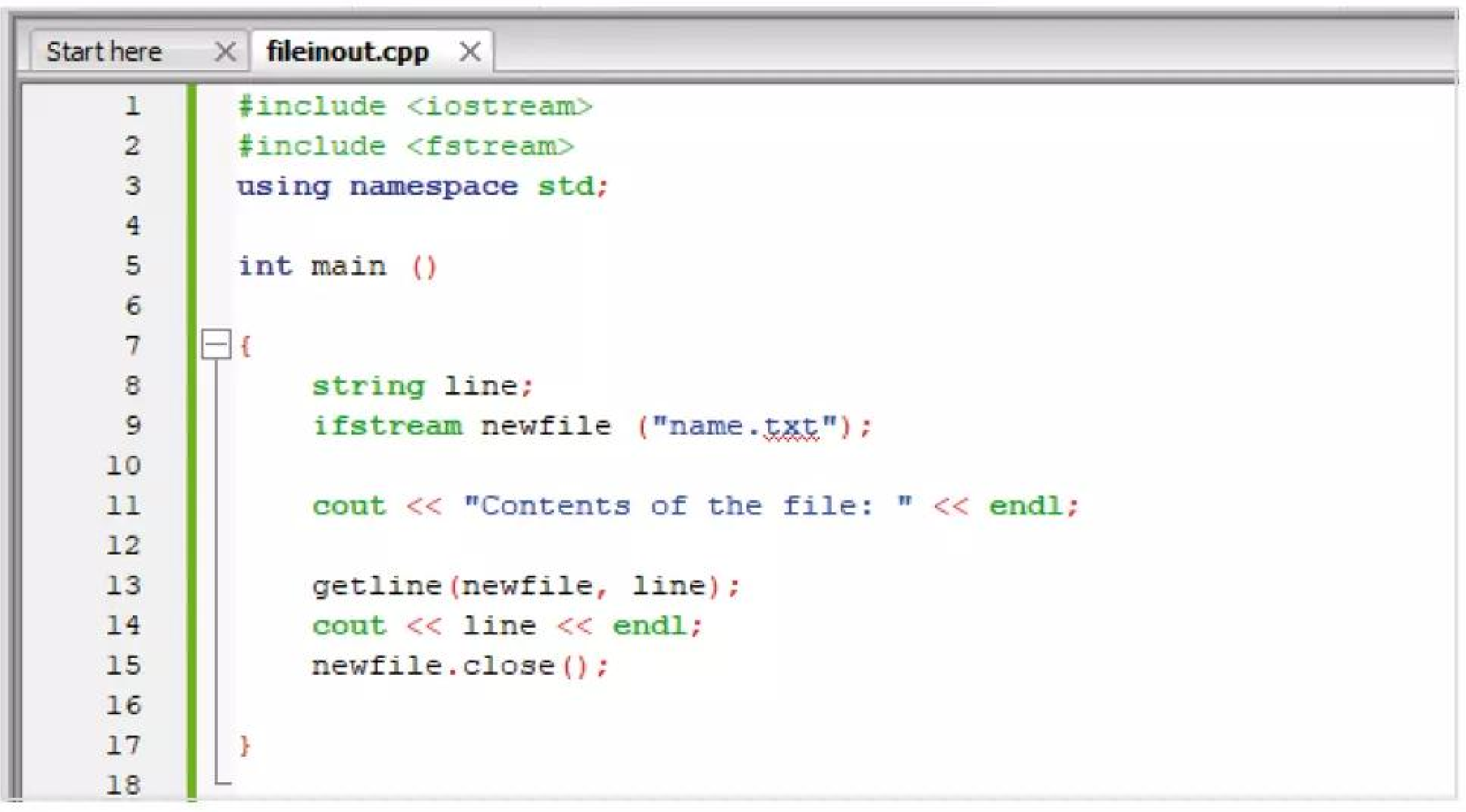
```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string name; // Create string to hold user's name

    ofstream newfile; // Create a new text file called name.txt
    newfile.open("name.txt");

    cout << "Enter your name: " << endl; // Ask user for their name and store the input
    getline(cin, name);

    newfile << name << endl; // Write stored input (name) to the text file

}
```

```
David Hayward
```

```
Enter your name:
```

STEP 4 To read the contents of a file, and output it to the screen, you need to do things slightly differently. First you need to create a string variable to store the file's contents (line by line), then open the file, use getline to read the file line by line and output those lines to the screen. Finally, close the file.

```
    string line;
    ifstream newfile ("name.txt");

    cout << "Contents of the file: " << endl;

    getline(newfile, line);
    cout << line << endl;
    newfile.close();
```

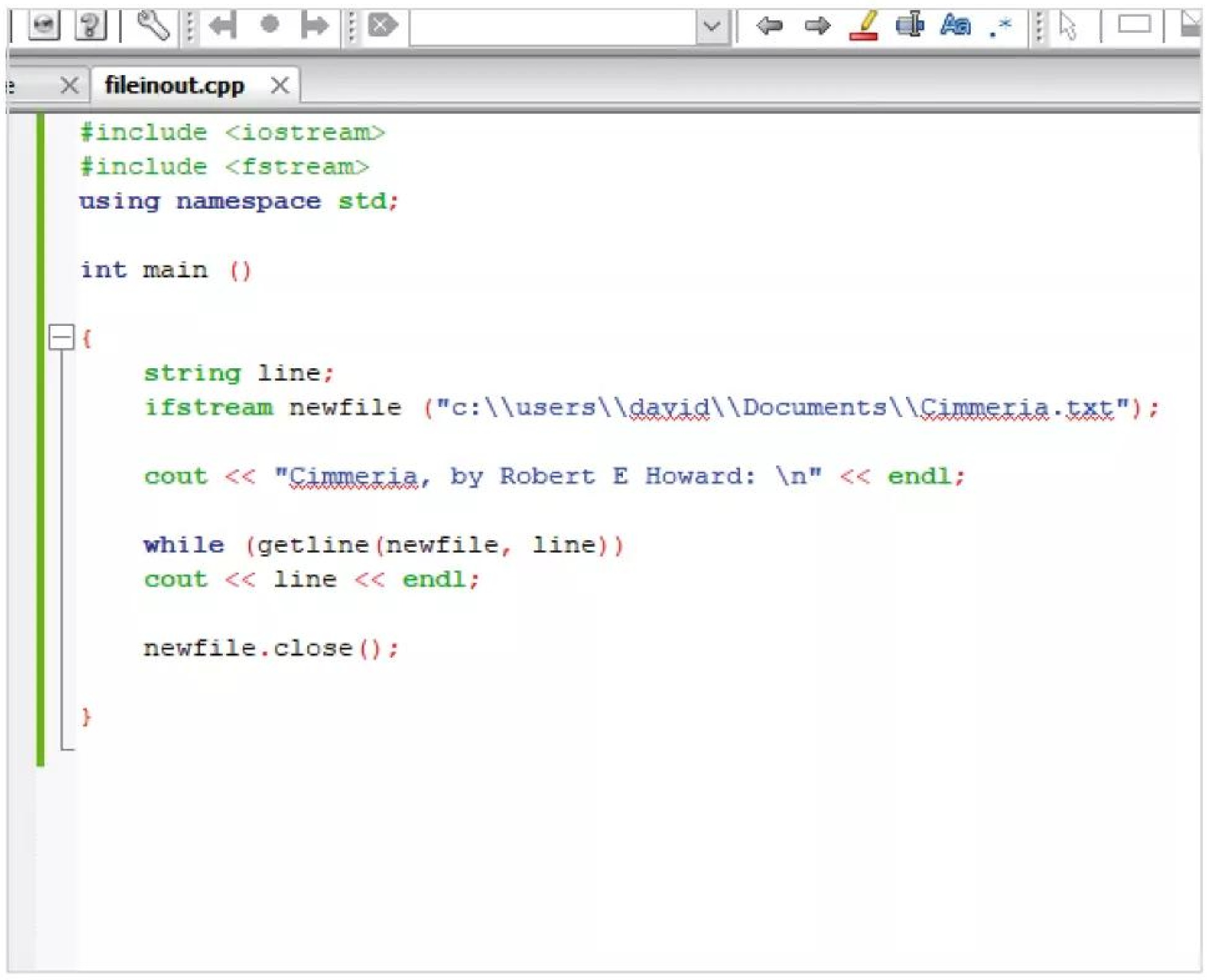


```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{
    string line;
    ifstream newfile ("name.txt");

    cout << "Contents of the file: " << endl;

    getline(newfile, line);
    cout << line << endl;
    newfile.close();

}

```

STEP 5

The code above is great for opening a file with one or two lines but what if there are multiple lines? Here we opened a text file of the poem Cimmeria, by Robert E Howard:

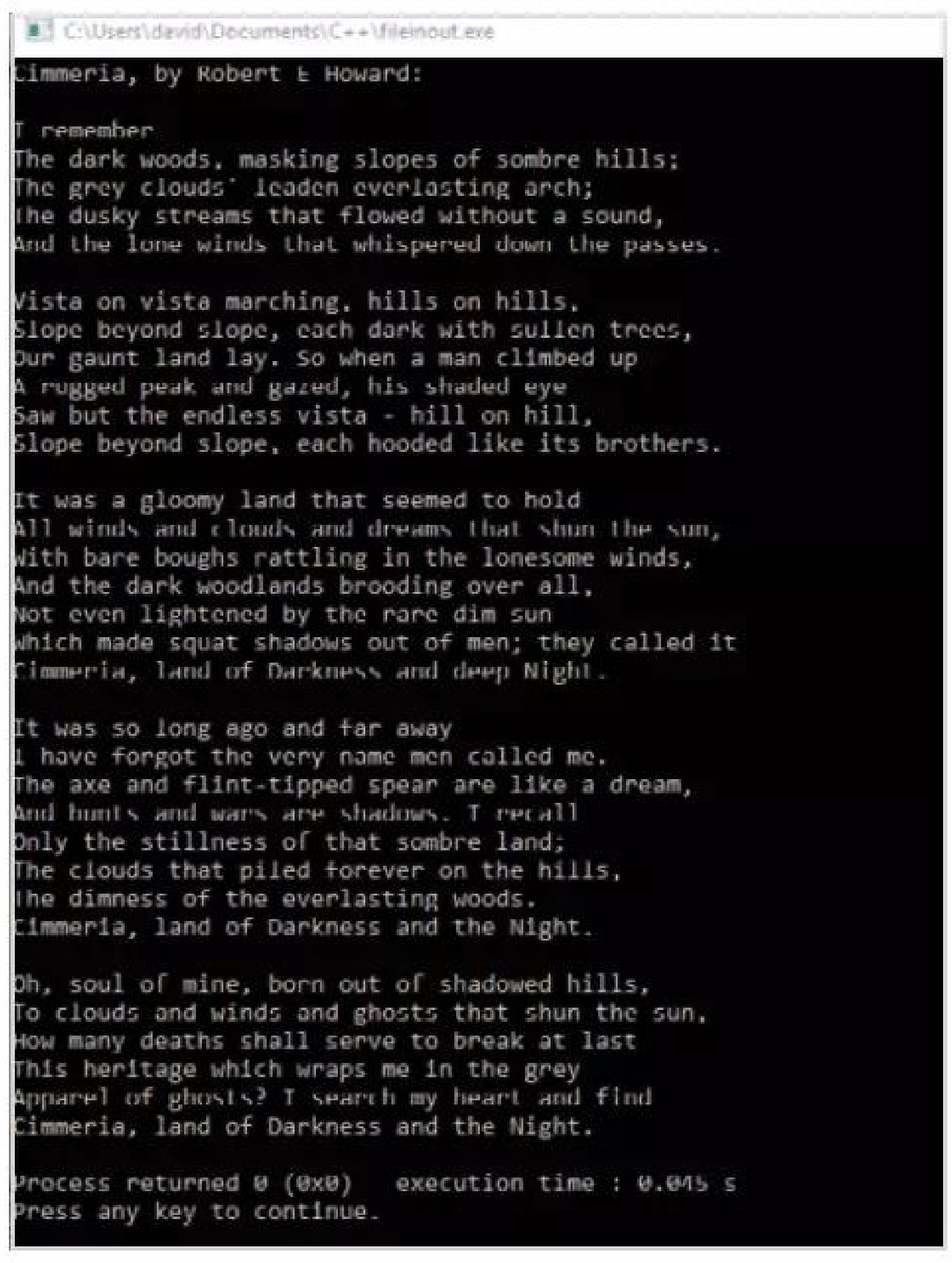
```
    string line;
    ifstream newfile (“c:\\users\\david\\
Documents\\Cimmeria.txt”);

    cout << “Cimmeria, by Robert E Howard: \n” <<
endl;

    while (getline(newfile, line))
    cout << line << endl;

    newfile.close();
```



```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string line;
    ifstream newfile ("c:\\users\\david\\Documents\\Cimmeria.txt");

    cout << "Cimmeria, by Robert E Howard: \n" << endl;

    while (getline(newfile, line))
    cout << line << endl;

    newfile.close();

}
```

STEP 6

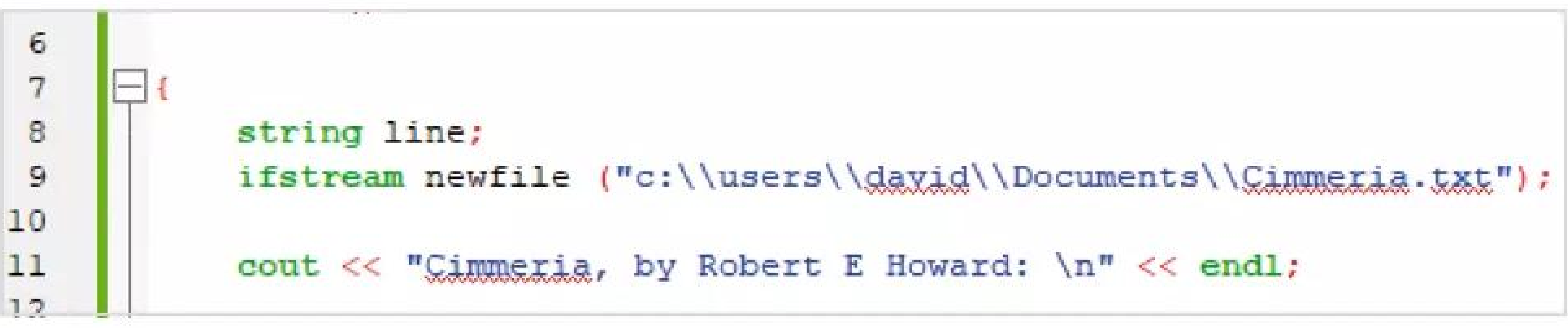
You can no doubt see that we’ve included a while loop, which we cover in a few pages time. It means that while there are lines to be read from the text file, C++ getlines them. Once all the lines are read, the output is displayed on the screen and the file is closed.



```
Cimmeria, by Robert E Howard:

I remember
The dark woods, masking slopes of sombre hills;
The grey clouds' leaden everlasting arch;
The dusky streams that flowed without a sound,
And the lone winds that whispered down the passes.

Vista on vista marching, hills on hills,
Slope beyond slope, each dark with sullen trees,
Our gaunt land lay. So when a man climbed up
A rugged peak and gazed, his shaded eye
Saw but the endless vista - hill on hill,
Slope beyond slope, each hooded like its brothers.

It was a gloomy land that seemed to hold
All winds and clouds and dreams that shun the sun,
With bare boughs rattling in the lonesome winds,
And the dark woodlands brooding over all,
Not even lightened by the rare dim sun
Which made squat shadows out of men; they called it
Cimmeria, land of Darkness and deep Night.

It was so long ago and far away
I have forgot the very name men called me.
The axe and flint-tipped spear are like a dream,
And hunts and wars are shadows. I recall
Only the stillness of that sombre land;
The clouds that piled forever on the hills,
The dimness of the everlasting woods.
Cimmeria, land of Darkness and the Night.

Oh, soul of mine, born out of shadowed hills,
To clouds and winds and ghosts that shun the sun,
How many deaths shall serve to break at last
This heritage which wraps me in the grey
Apparel of ghosts? I search my heart and find
Cimmeria, land of Darkness and the Night.

Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

STEP 7

You can also see that the location of the text file Cimmeria.txt isn’t in the same folder as the C++ program. When we created the first name.txt file, it was written to the same folder where the code was located; this is done by default. To specify another folder, you need to use double-back slashes, as per the character literals/escape sequence code.

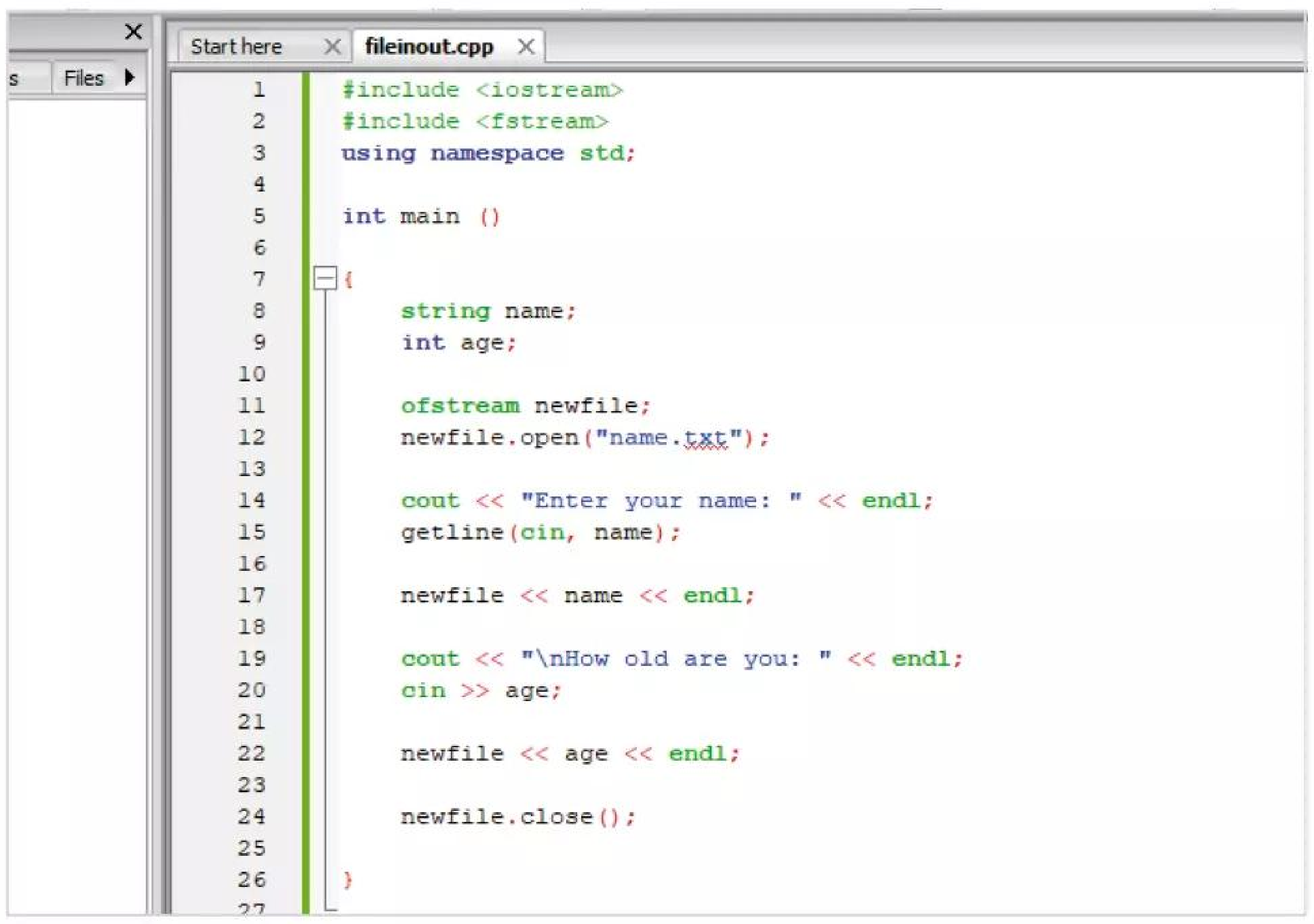


```
{
    string line;
    ifstream newfile ("c:\\users\\david\\Documents\\Cimmeria.txt");

    cout << "Cimmeria, by Robert E Howard: \n" << endl;
```

STEP 8

Just as you might expect, you can write almost anything you like to a file, for reading either in Notepad or via the console through the C++ code:

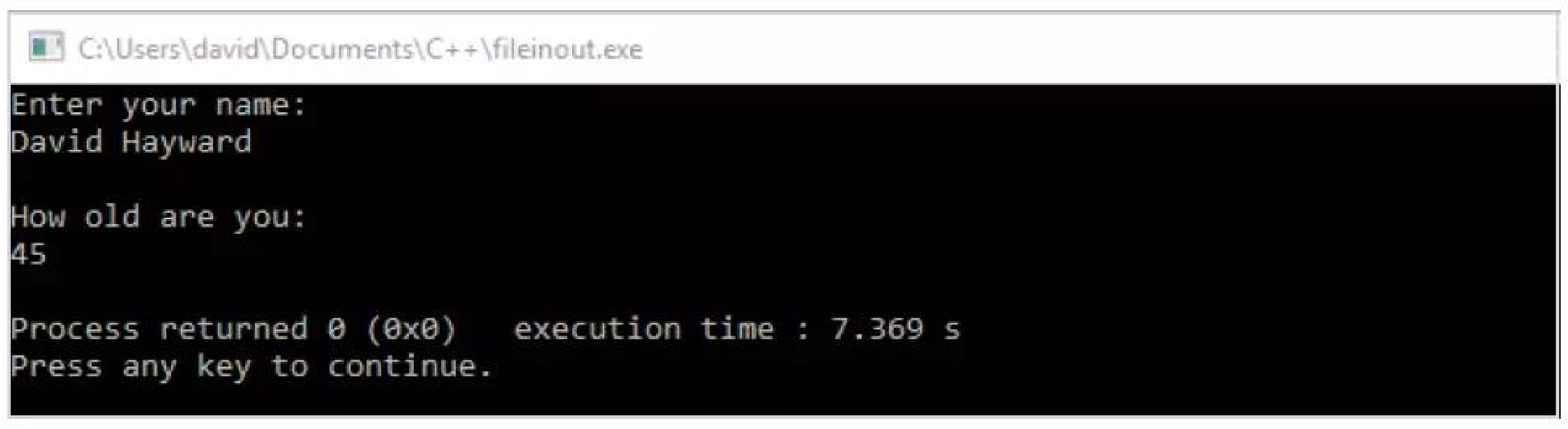
```
    string name;
    int age;

    ofstream newfile;
    newfile.open(“name.txt”);

    cout << “Enter your name: “ << endl;
    getline(cin, name);

    newfile << name << endl;

    cout << “\nHow old are you: “ << endl;
    cin >> age;

    newfile << age << endl;

    newfile.close();
```



```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

    string name;
    int age;

    ofstream newfile;
    newfile.open("name.txt");

    cout << "Enter your name: " << endl;
    getline(cin, name);

    newfile << name << endl;

    cout << "\nHow old are you: " << endl;
    cin >> age;

    newfile << age << endl;

    newfile.close();

}
```

STEP 9

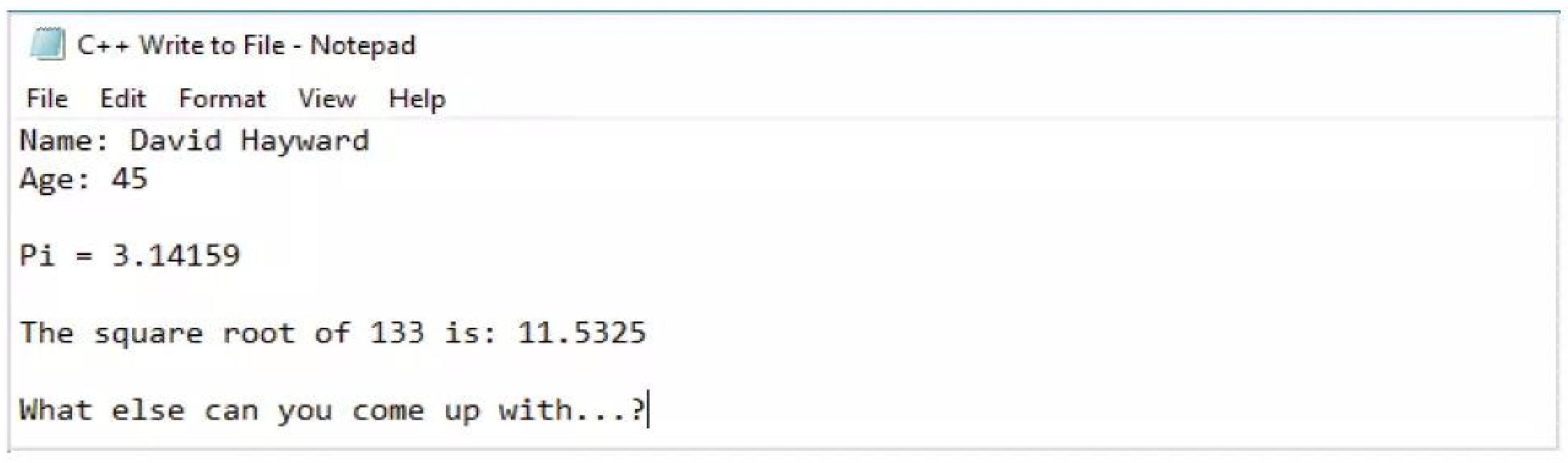
The code from step 8 differs again, but only where it comes to adding the age integer. Notice that we used cin >> age, instead of the previous getline(cin, variable). The reason for this is that the getline function handles strings, not integers; so when you’re using a data type other than a string, use the standard cin.



```
Enter your name:
David Hayward

How old are you:
45

Process returned 0 (0x0)   execution time : 7.369 s
Press any key to continue.
```

STEP 10

Here’s an exercise: see if you can create code to write several different elements to a text file. You can have a user’s name, age, phone number etc. Maybe even the value of Pi and various mathematical elements. It’s all good practice.



```
Name: David Hayward
Age: 45

Pi = 3.14159

The square root of 133 is: 11.5325

What else can you come up with...?
```

Loops and Decision Making

Loops and repetition are one of the most important factors of any programming language. Good use of a loop creates a program that does exactly what you want it to and delivers the desired outcome without issues or errors.

Without loops and decision making events within the code, your program will never be able to offer the user any choice. It's this understanding of choice that elevates your skills as a programmer and makes for much better code.

While Loop

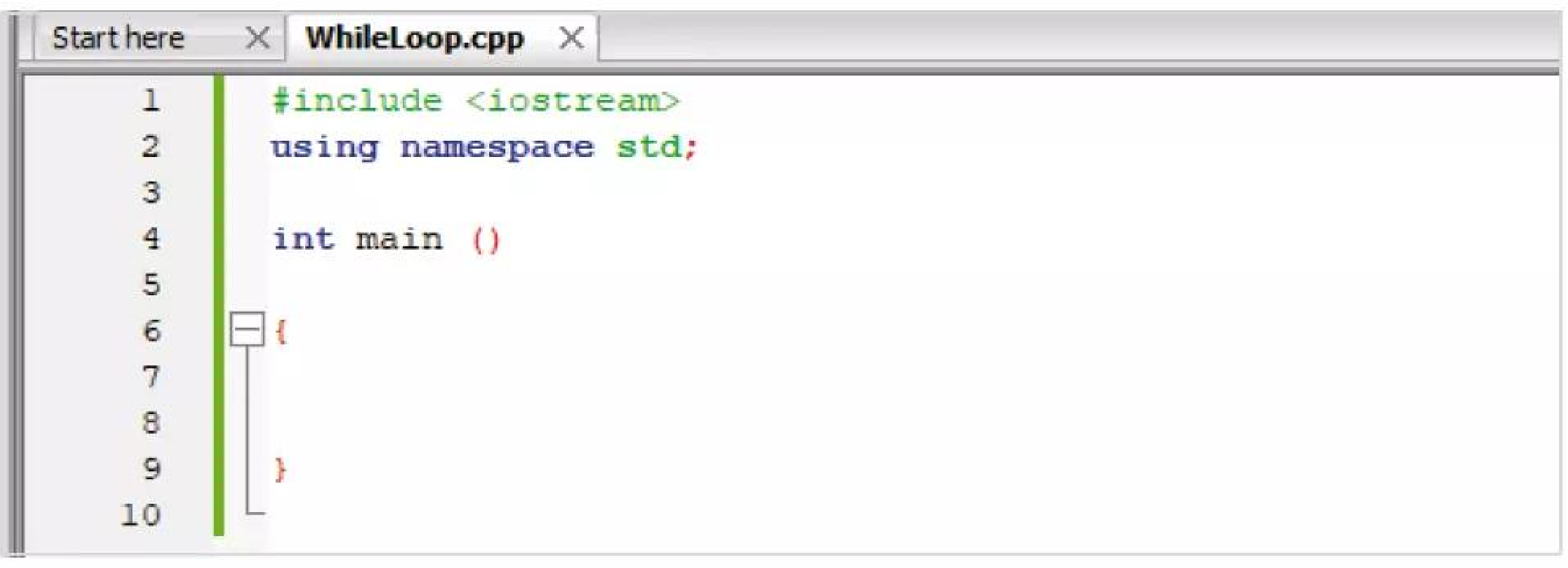
A while loop's function is to repeat a statement, or a group of statements, while a certain condition remains true. When the while loop starts, it initialises itself by testing the condition of the loop and the statements within, before executing the rest of the loop.

TRUE OR FALSE?

While loops are one of the most popular form of C++ code looping. They repeatedly run the code contained within the loop while the condition is true. Once it proves false, the code continues as normal.

STEP 1 Clear what you've done so far and create a new C++ file. There's no need for any extra headers at the moment, so add the standard headers as per usual:

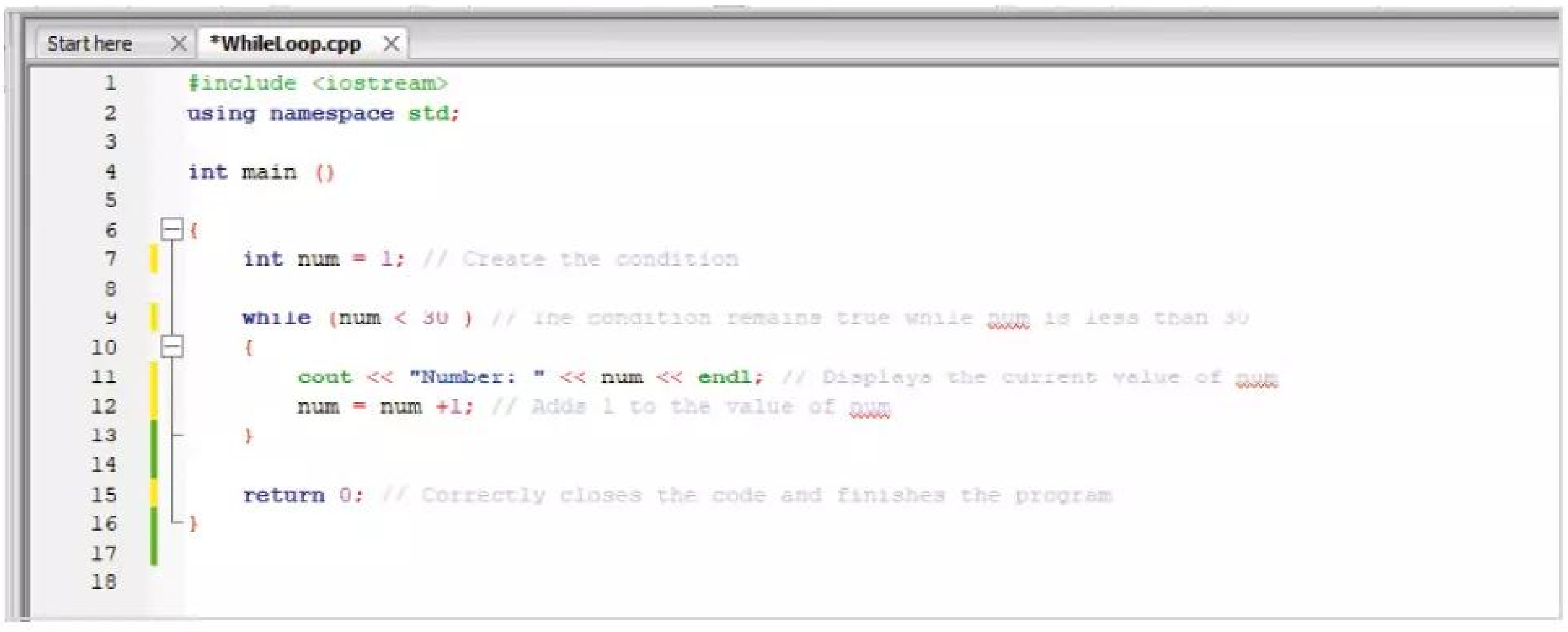
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{


}
```

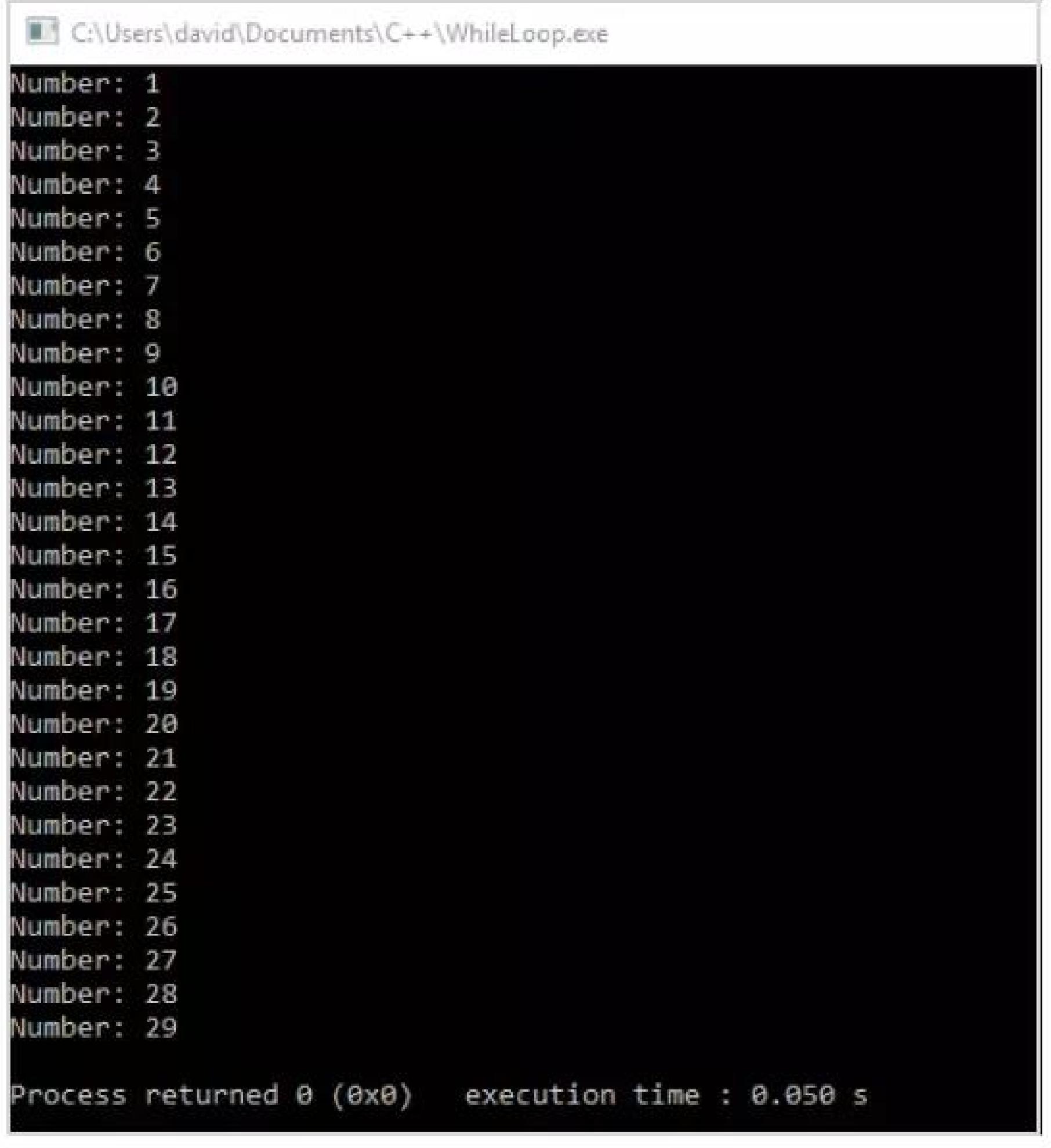
STEP 2 Create a simple while loop. Enter the code below, build and run (we've added comments to the screen shot):

```
{
    int num = 1;

    while (num < 30 )
    {
        cout << "Number: " << num << endl;
        num = num +1;
    }

    return 0;
}
```



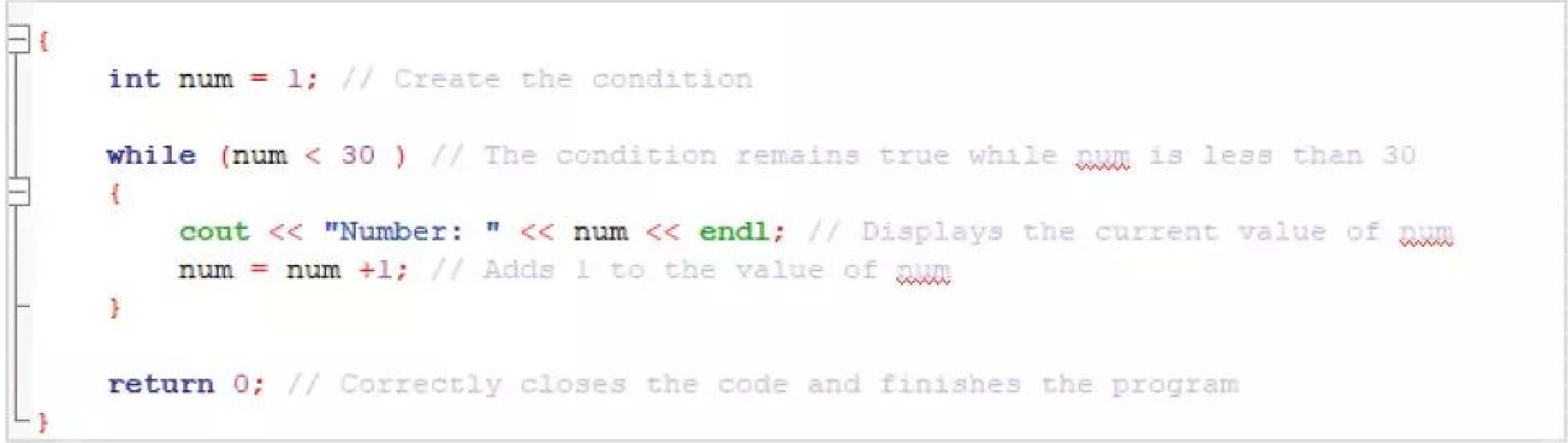
```
#include <iostream>
using namespace std;

int main ()

{
    int num = 1; // Create the condition

    while (num < 30 ) // The condition remains true while num is less than 30
    {
        cout << "Number: " << num << endl; // Displays the current value of num
        num = num +1; // Adds 1 to the value of num
    }

    return 0; // Correctly closes the code and finishes the program
}


```

STEP 3 First you need to create a condition, so use a variable called num and give it the value 1. Now create the while loop, stating that as long as num is less than 30, the loop is true. Within the loop the value of num is displayed and adds 1 until it's more than 30.



```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Number: 11
Number: 12
Number: 13
Number: 14
Number: 15
Number: 16
Number: 17
Number: 18
Number: 19
Number: 20
Number: 21
Number: 22
Number: 23
Number: 24
Number: 25
Number: 26
Number: 27
Number: 28
Number: 29

Process returned 0 (0x0)   execution time : 0.050 s
```

STEP 4 We're introducing a few new elements here. The first are the opening and closing braces for the while loop. This is because our loop is a compound statement, meaning a group of statements; note also, there's no semicolon after the while statement. You now also have return 0, which is a clean and preferred way of ending the code.



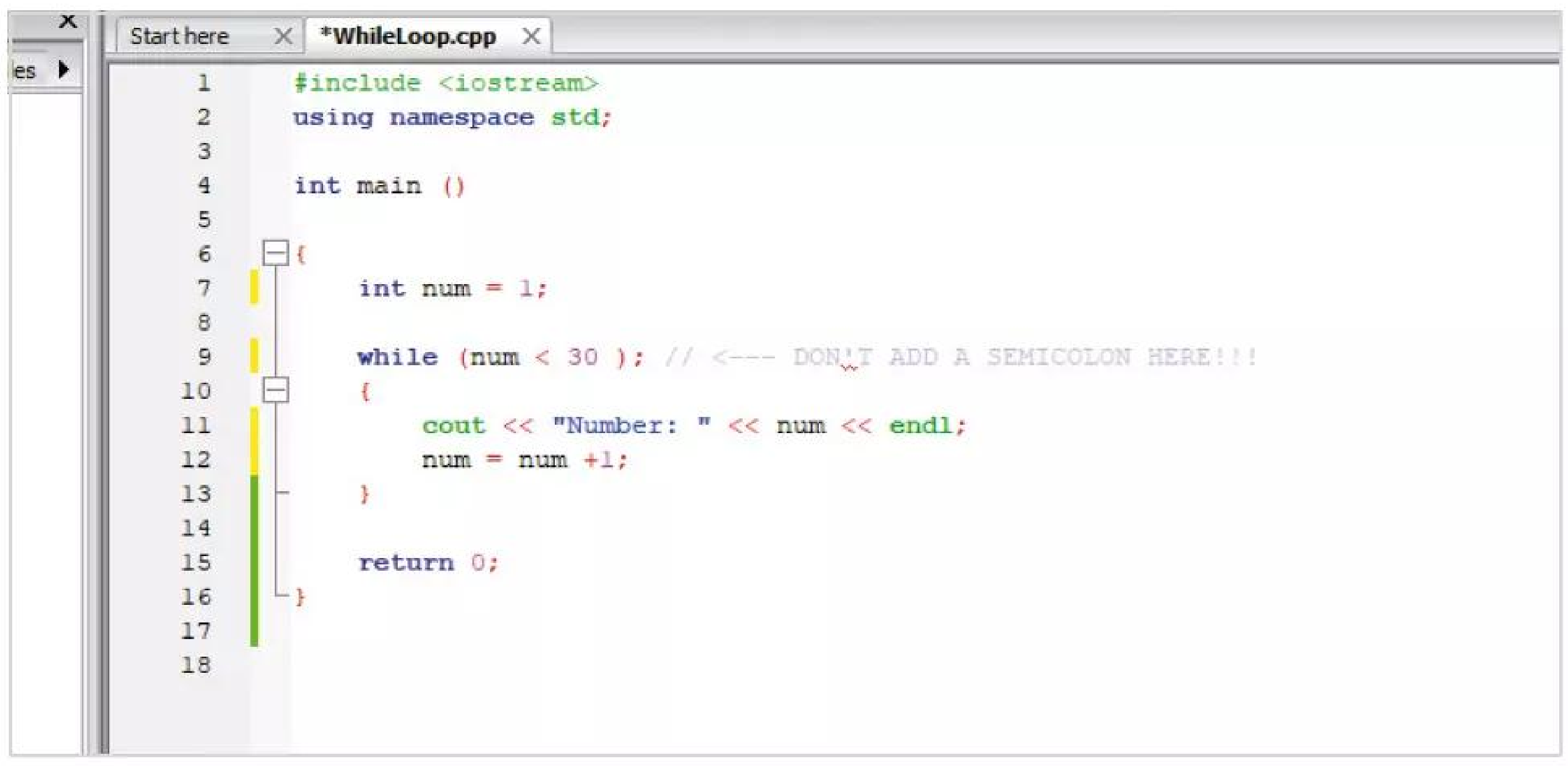
```
{
    int num = 1; // Create the condition

    while (num < 30 ) // The condition remains true while num is less than 30
    {
        cout << "Number: " << num << endl; // Displays the current value of num
        num = num +1; // Adds 1 to the value of num
    }

    return 0; // Correctly closes the code and finishes the program
}
```

STEP 5 If you didn't need to see the continually increasing value of num, you could have done away with the compound while statement and instead just added num by itself until it reached 30, and then displayed the value:

```
{
    int num = 1;

    while (num < 30 )
        num++;
    cout << "Number: " << num << endl;

    return 0;
}
```

STEP 6

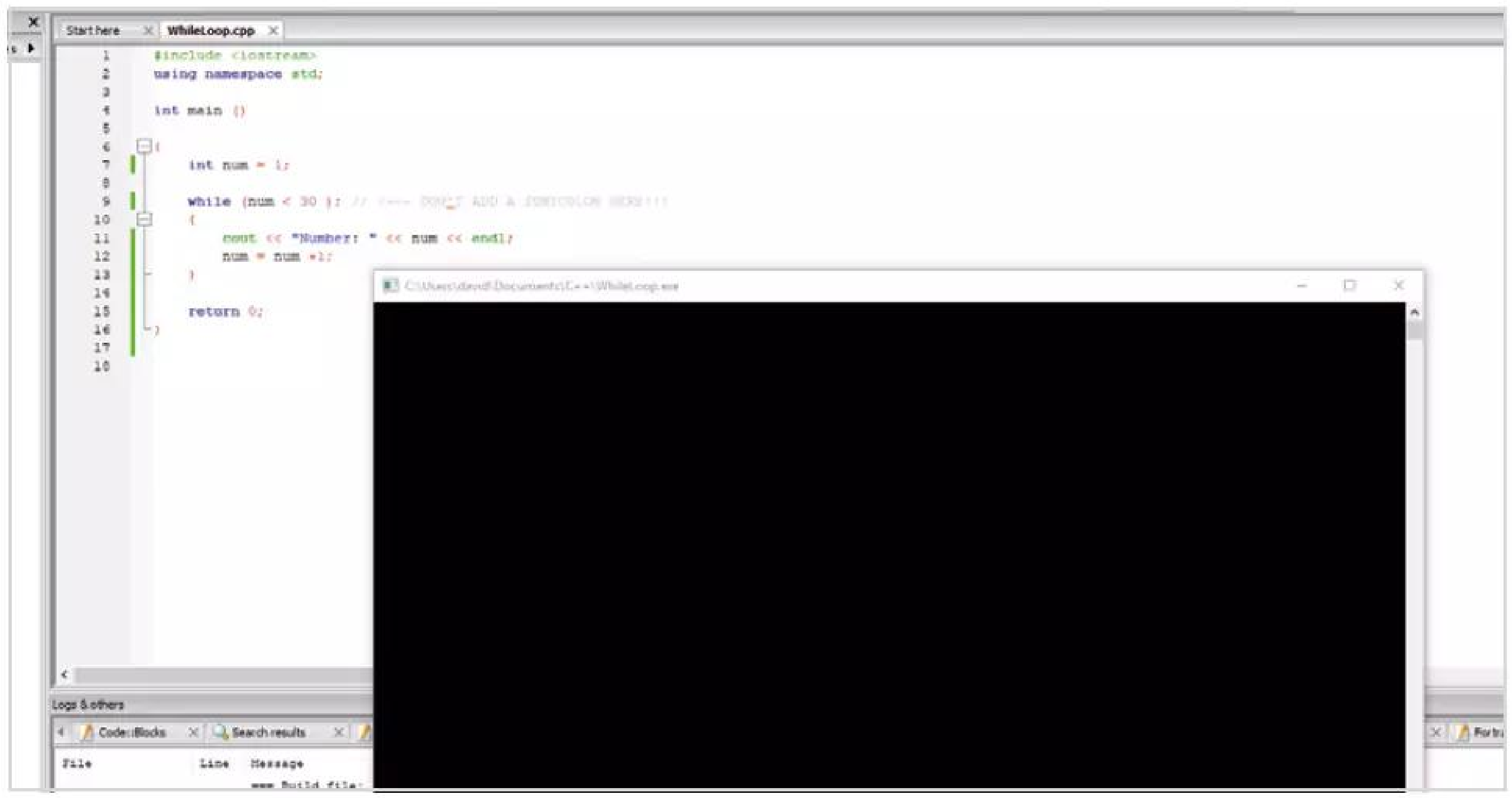
It's important to remember not to add a semicolon at the end of a while statement. Why? Well, as you know, the semicolon represents the end of a C++ line of code. If you place one at the end of a while statement, your loop will be permanently stuck until you close the program.



```
#include <iostream>
using namespace std;

int main ()

{

    int num = 1;

    while (num < 30 ); // <--- DON'T ADD A SEMICOLON HERE!!!
    {
        cout << "Number: " << num << endl;
        num = num +1;
    }

    return 0;
}
```

STEP 7

In our example, if we were to execute the code the value of num would be 1, as set by the int statement. When the code hits the while statement it reads that while the condition of 1 being less than 30 is true, loop. The semicolon closes the line, so the loop repeats; but it never adds 1 to num, as it won't continue through the compound statement.



```
#include <iostream>
using namespace std;

int main ()

{

    int num = 1;

    while (num < 30 ); // <--- DON'T ADD A SEMICOLON HERE!!!
    {
        cout << "Number: " << num << endl;
        num = num +1;
    }

    return 0;
}
```

STEP 8

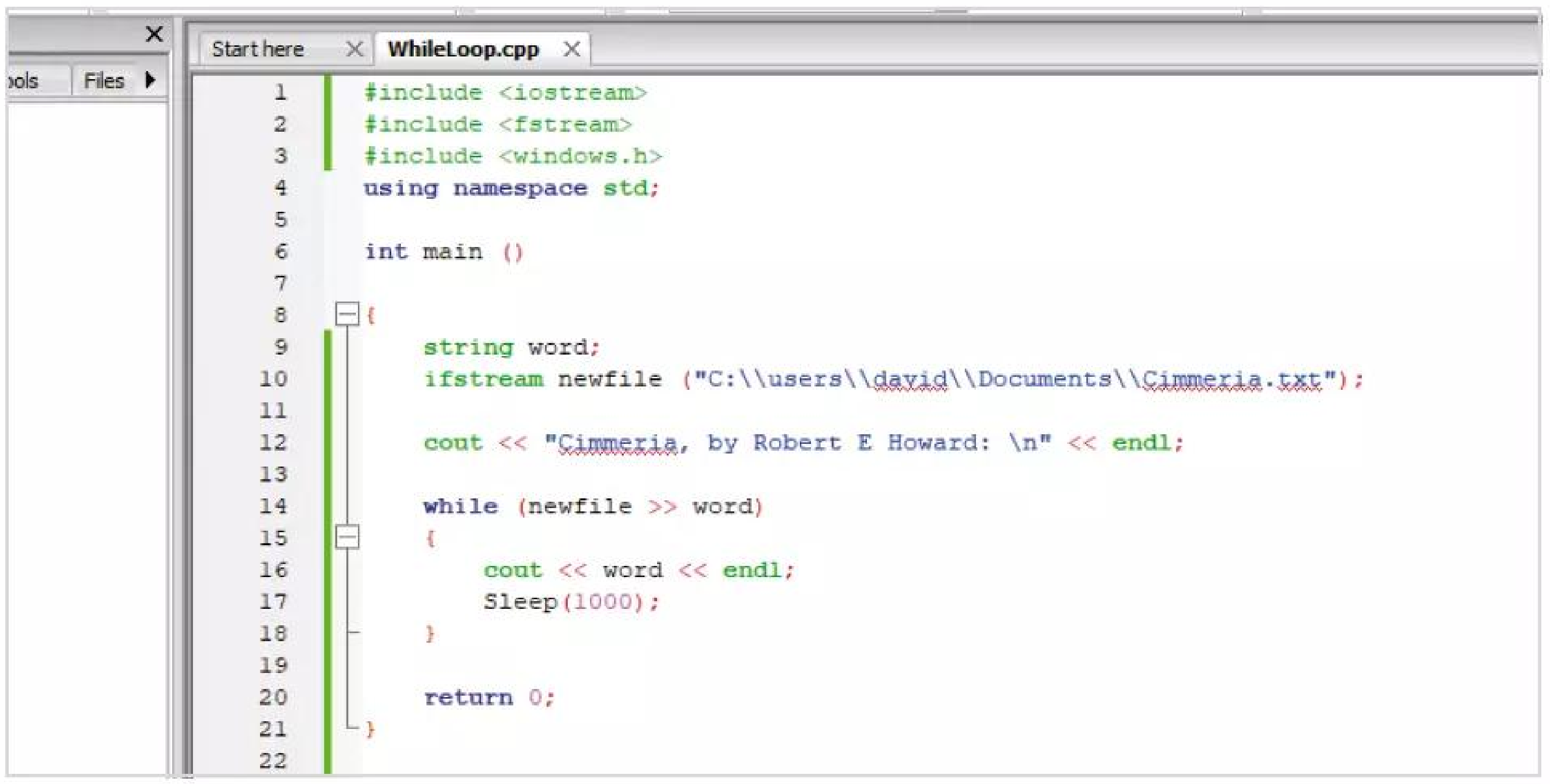
You can manipulate the while statement to display different results depending on what code lies within the loop. For example, to read the poem, Cimmeria, word by word, you would enter:

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{
    string word;
    ifstream newfile (“C:\\users\\david\\
Documents\\Cimmeria.txt”);

    cout << “Cimmeria, by Robert E Howard: \n” <<
endl;

    while (newfile >> word)
    {
        cout << word << endl;
    }

    return 0;
}
```

STEP 9

You can further expand the code to enable each word of the poem to appear every second. To do so, you need to pull in a new library, <windows.h>. This is a Windows only library and within it you can use the Sleep() function:

```
#include <iostream>
#include <fstream>
#include <windows.h>
using namespace std;

int main ()

{
    string word;
    ifstream newfile (“C:\\users\\david\\
Documents\\Cimmeria.txt”);

    cout << “Cimmeria, by Robert E Howard: \n” <<
endl;

    while (newfile >> word)
    {
        cout << word << endl;
        Sleep(1000);
    }

    return 0;
}
```



```
#include <iostream>
#include <fstream>
#include <windows.h>
using namespace std;

int main ()

{
    string word;
    ifstream newfile ("C:\\users\\david\\Documents\\Cimmeria.txt");

    cout << "Cimmeria, by Robert E Howard: \n" << endl;

    while (newfile >> word)
    {
        cout << word << endl;
        Sleep(1000);
    }

    return 0;
}
```

STEP 10

Sleep() works in milliseconds, so Sleep(1000) is one second, Sleep(10000) is ten seconds and so on. Combining the sleep function (along with the header it needs) and a while loop enables you to come up with some interesting countdown code.

```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    int a = 10;

    while (a != 0)
    {
        cout << a << endl;
        a = a - 1;
        Sleep(1000);
    }

    cout << “\nBlast Off!” << endl;

    return 0;
}
```

For Loop

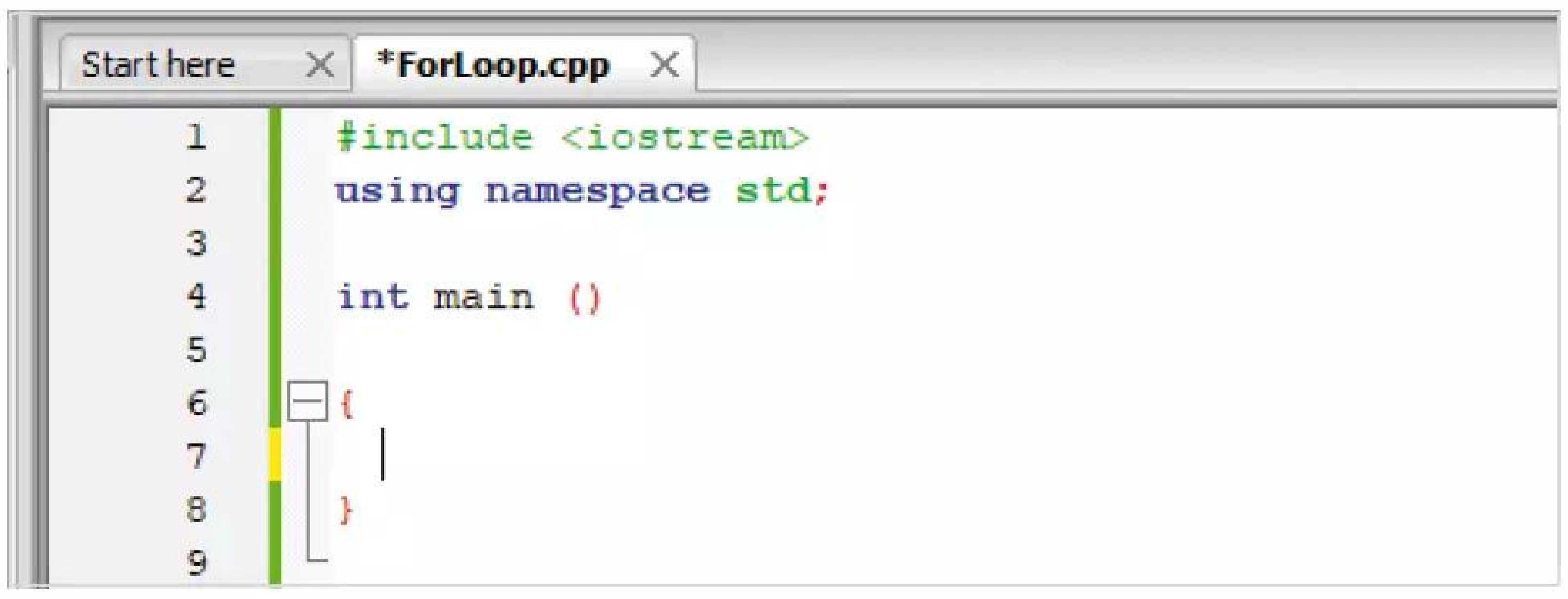
In some respects, a for loop works in a very similar way to that of a while loop, although it's structure is different. A for loop is split into three stages: an initialiser, a condition and an incremental step. Once set up, the loop repeats itself until the condition becomes false.

LOOPY LOOPS

The initialise stage of a for loop is executed only once and this sets the point reference for the loop. The condition is evaluated by the loop to see if it's true or false and then the increment is executed. The loop then repeats the second and third stage.

STEP 1 Create a new C++ file, with the standard headers:

```
#include <iostream>
using namespace std;

int main ()

{


}
```

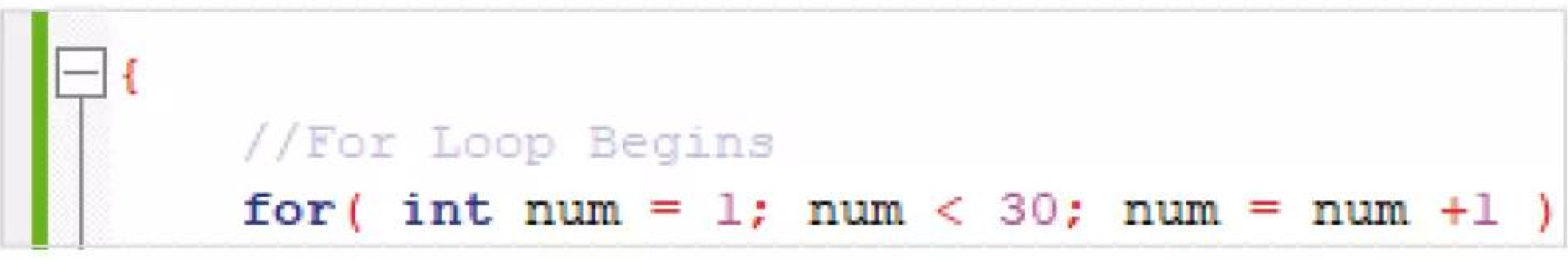


```
#include <iostream>
using namespace std;

int main ()

{
    |
}
```

STEP 2 Start simple and create a for loop that counts from 1 to 30, displaying the value to the screen with each increment:

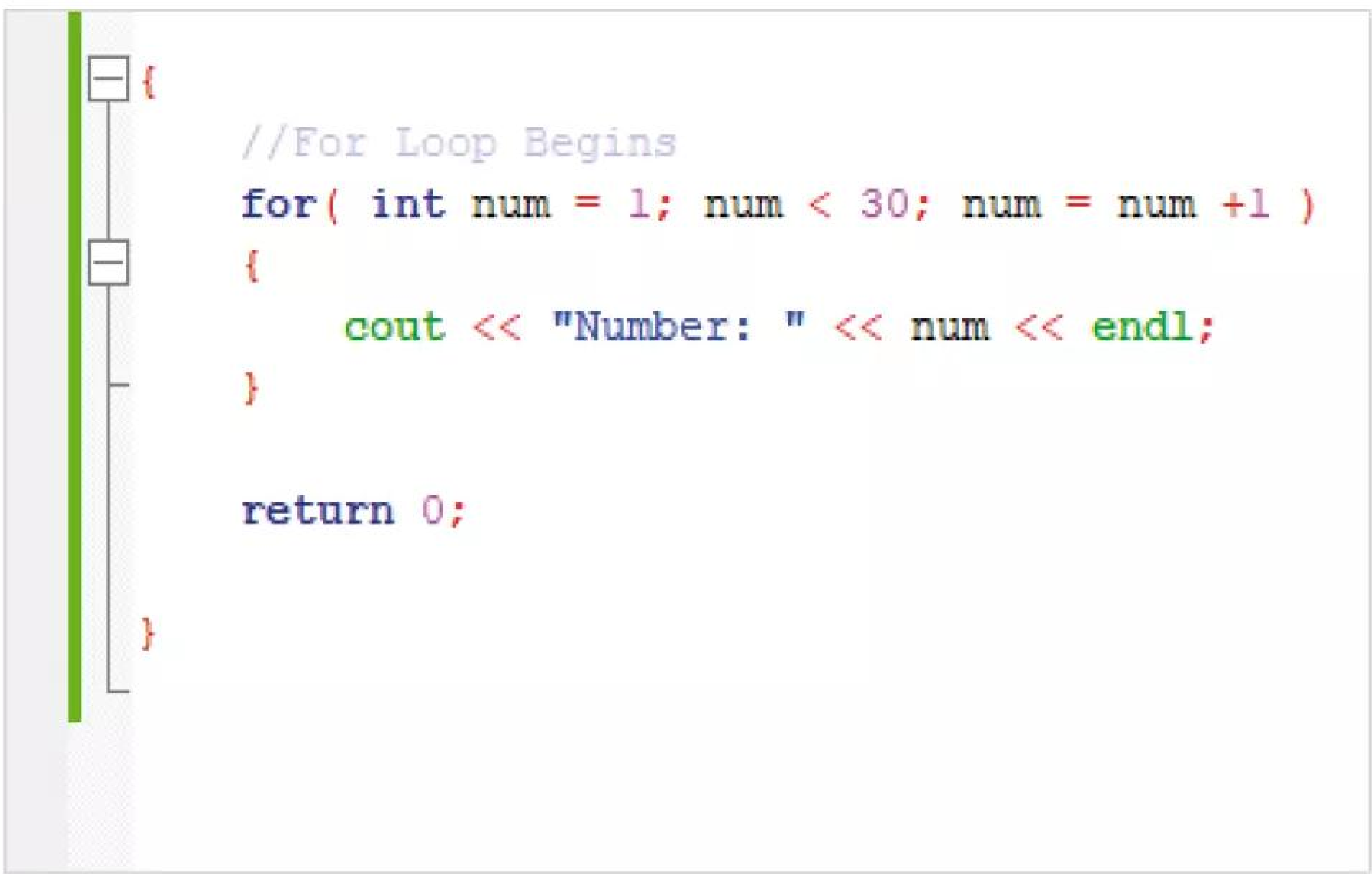
```
{
    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
    {
        cout << “Number: “ << num << endl;
    }

    return 0;

}
```

STEP 3 Working through the process of the for loop, begin by creating an integer called num and assigning it a value of 1. Next, set the condition, in this case num being less than 30. The last stage is where you create the increments; here it's the value of num being added by 1.



```
{

    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
```

STEP 4 After the loop, you created a compound statement in braces (curly brackets), that displays the current value of the integer num. Every time the for loop repeats itself, the second and third stages of the loop, it adds 1 until the condition <30 is false. The loop then ends and the code continues, ending neatly with return 0.



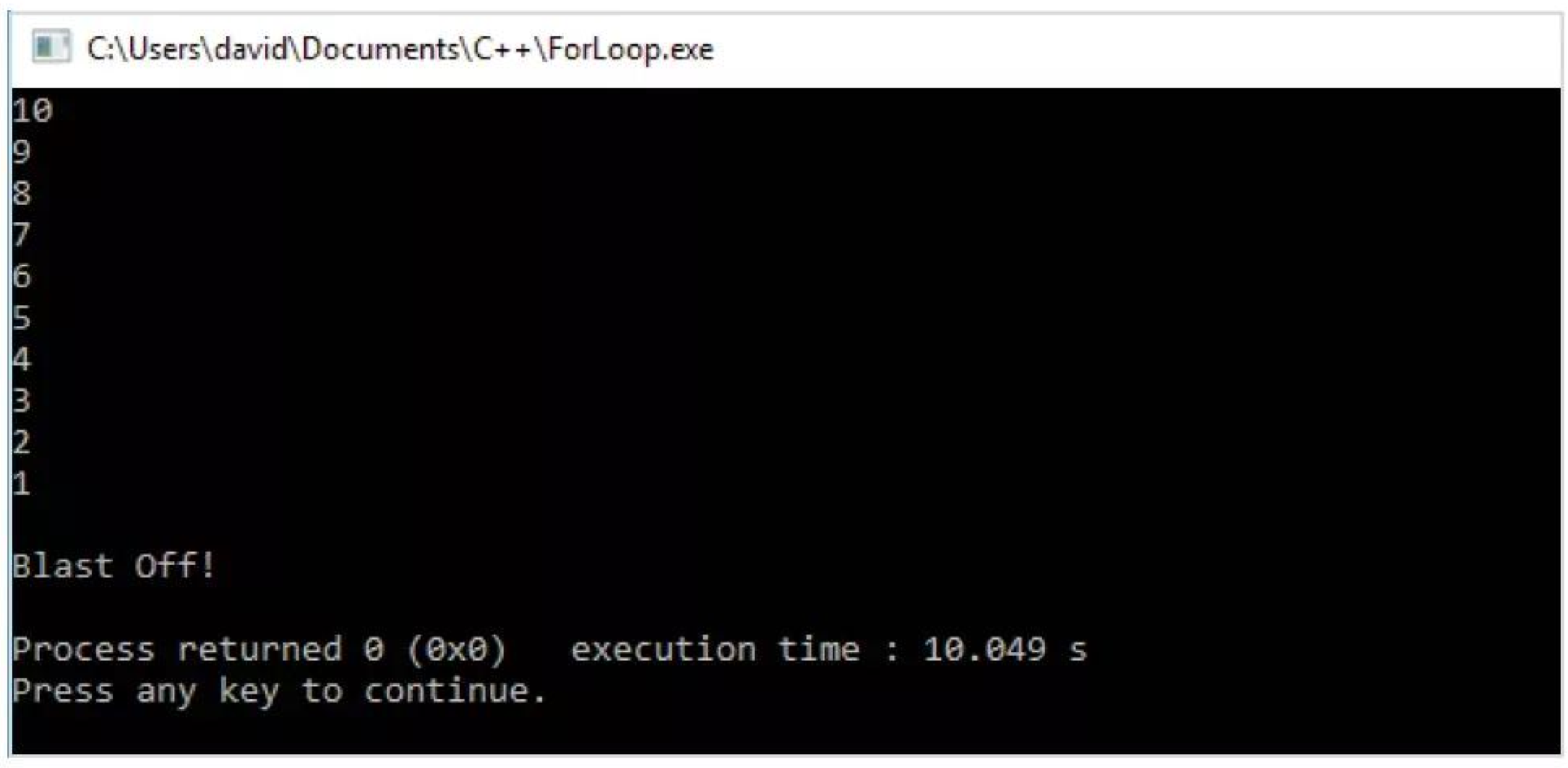
```
{
    //For Loop Begins
    for( int num = 1; num < 30; num = num +1 )
    {
        cout << "Number: " << num << endl;
    }

    return 0;

}
```

STEP 5 A for loop is quite a neat package in C++, all contained within its own brackets, while the other elements outside of the loop are displayed below. If you want to create a 10-second countdown, you could use:

```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    //For Loop Begins
    for( int a = 10; a != 0; a = a -1 )
    {
        cout << a << endl;
        Sleep(1000);
    }

    cout << “\nBlast Off!” << endl;

    return 0;

}
```

STEP 6

With the countdown code, don't forget to include the windows.h library, so you can use the Sleep command. Build and run the code; in the command console you can see the numbers 10 to 1 countdown in one second increments, until it reaches zero and Blast Off! appears.

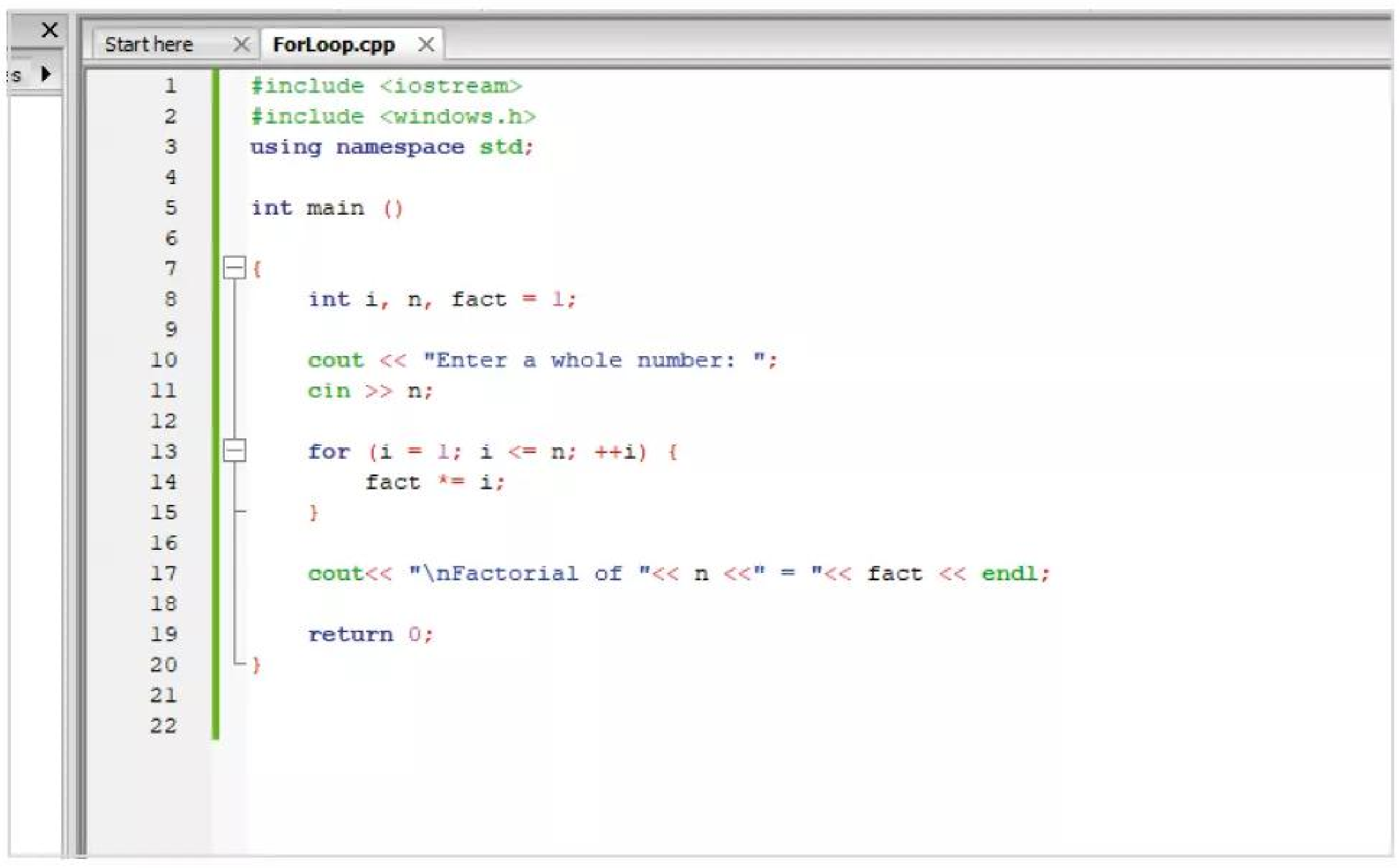


```
10
9
8
7
6
5
4
3
2
1

Blast Off!

Process returned 0 (0x0)   execution time : 10.049 s
Press any key to continue.
```

STEP 7

Naturally you can include a lot more content into a for loop, including some user input:

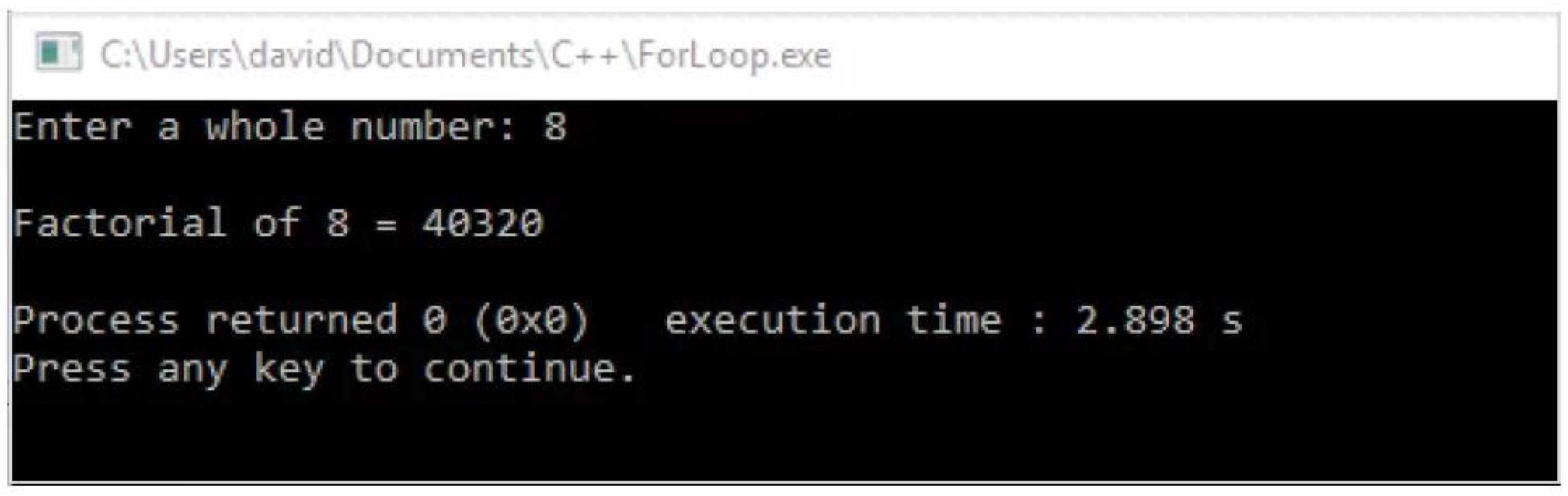
```
    int i, n, fact = 1;

    cout << “Enter a whole number: “;
    cin >> n;

    for (i = 1; i <= n; ++i) {
        fact *= i;
    }

    cout<< “\nFactorial of “<< n <<” = “<< fact <<
endl;

    return 0;
```



```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    int i, n, fact = 1;

    cout << "Enter a whole number: ";
    cin >> n;

    for (i = 1; i <= n; ++i) {
        fact *= i;
    }

    cout<< "\nFactorial of "<< n <<" = "<< fact << endl;

    return 0;
}
```

STEP 8

The code from step 7, when built and run, asks for a number, then displays the factorial of that number through the for loop. The user's number is stored in the integer n, followed by the integer I which is used to check if the condition is true or false, adding 1 each time and comparing it to the user's number, n.

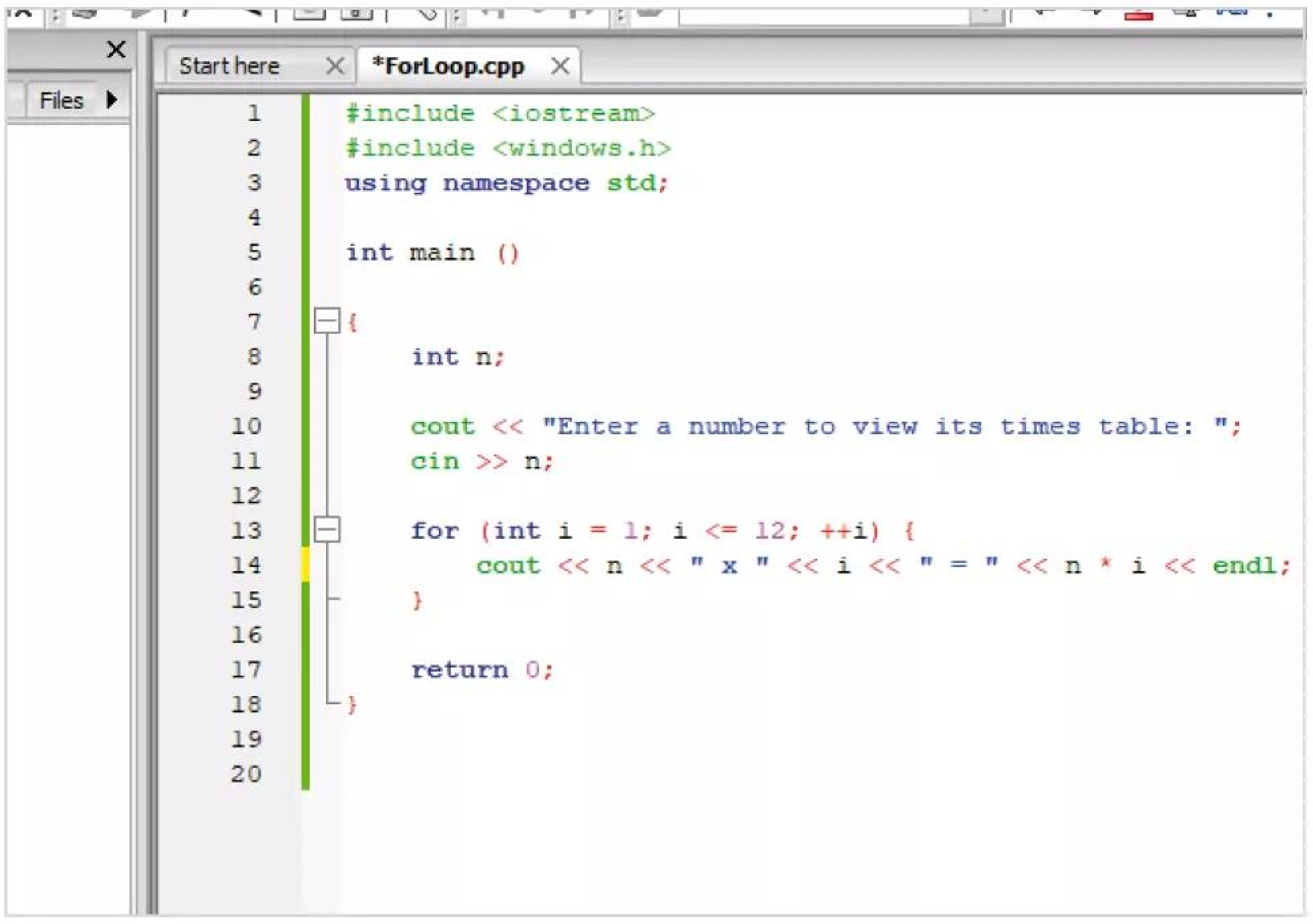


```
Enter a whole number: 8

Factorial of 8 = 40320

Process returned 0 (0x0)   execution time : 2.898 s
Press any key to continue.
```

STEP 9

Here's an example of a for loop displaying the multiplication tables of a user inputted number. Handy for students:

```
{
    int n;

    cout << “Enter a number to view its times
table: “;
    cin >> n;

    for (int i = 1; i <= 12; ++i) {
        cout << n << “ x “ << i << “ = “ << n * i
<< endl;
    }

    return 0;
}
```

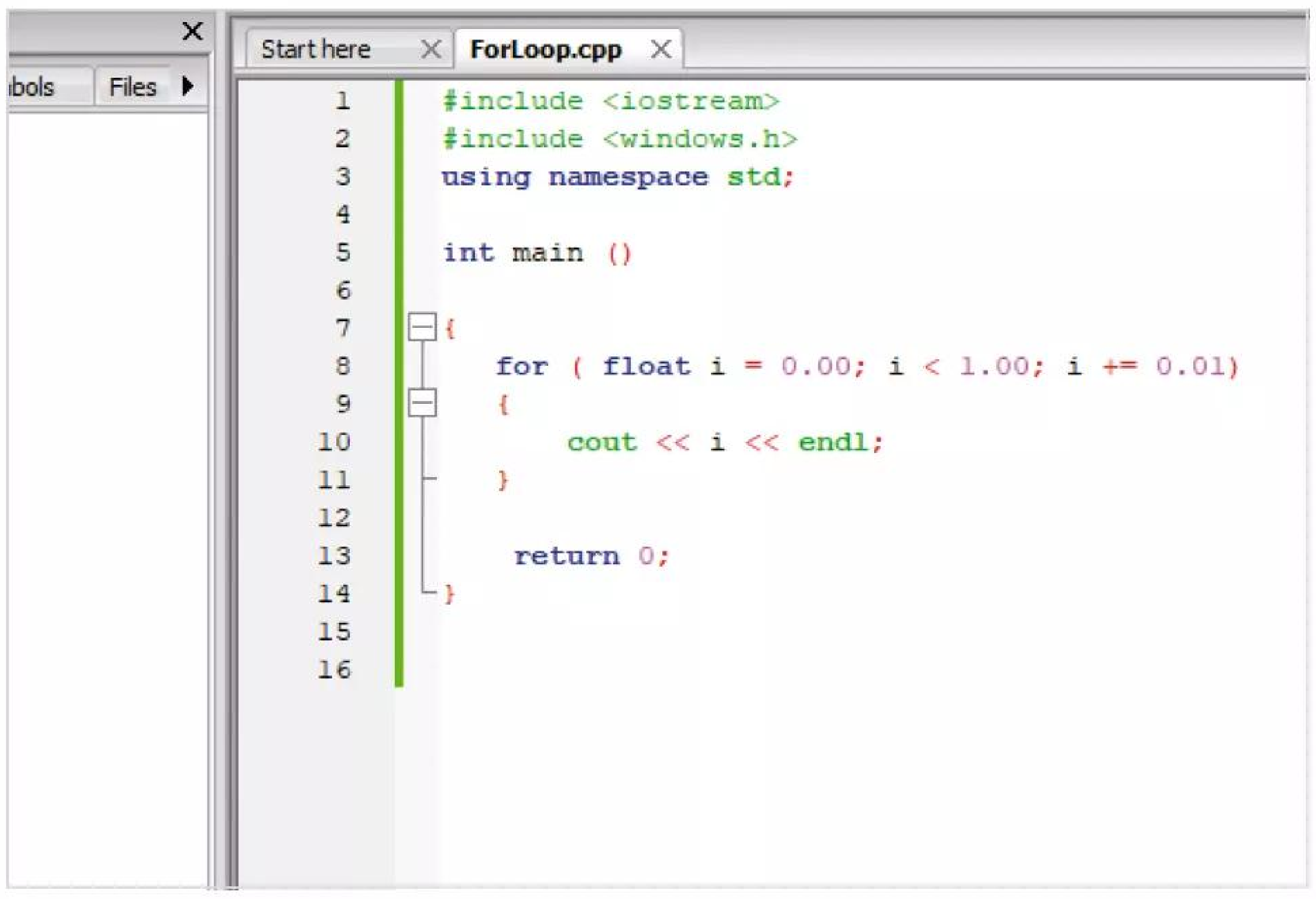


```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    int n;

    cout << "Enter a number to view its times table: ";
    cin >> n;

    for (int i = 1; i <= 12; ++i) {
        cout << n << " x " << i << " = " << n * i << endl;
    }

    return 0;
}
```

STEP 10

The value of the integer i can be expanded from 12 to whatever number you want, displaying a very large multiplication table in the process (or a small one). Of course the data type within a for loop doesn't have to be an integer; as long as it's valid, it works.

```
for ( float i = 0.00; i < 1.00; i += 0.01)
  {
      cout << i << endl;
  }
return 0;
```



```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()

{
    for ( float i = 0.00; i < 1.00; i += 0.01)
    {
        cout << i << endl;
    }

     return 0;
}
```

Do… While Loop

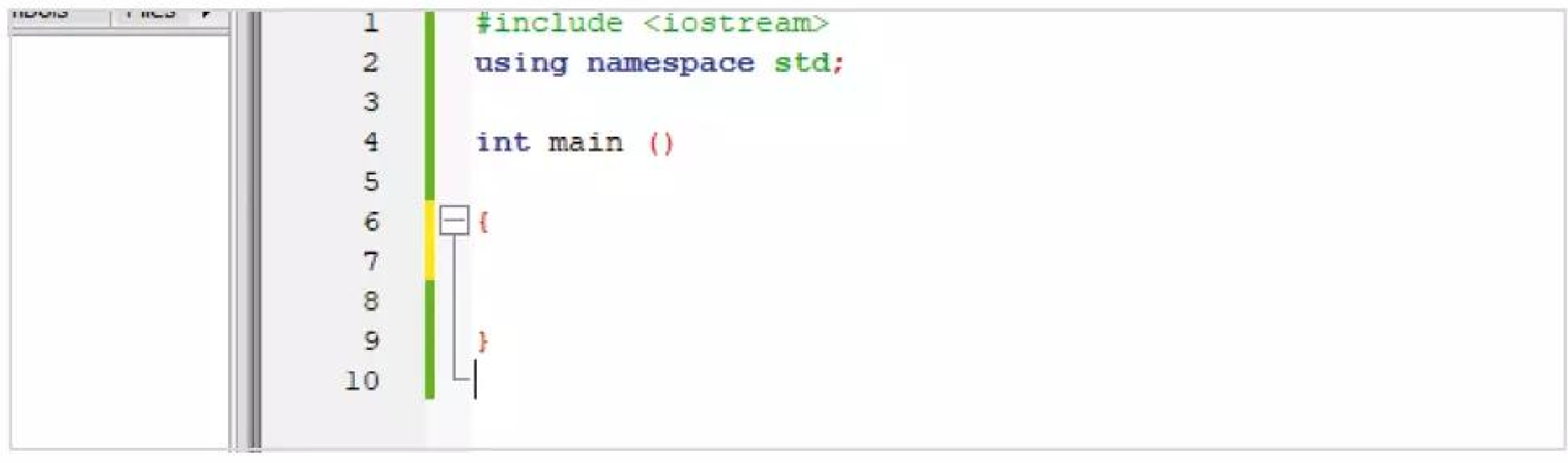
A do… while loop differs slightly from that of a for or even a while loop. Both for and while set and examine the state of the condition at the start of the loop, or the top of the loop if you prefer. However, a do… while loop, is similar to a while loop but instead checks the condition at the bottom of the loop.

DO LOOPS

The good thing about a do... while loop is that it's guaranteed to run through at least once. It's structure is: do, followed by statements, while condition is true. This is how it works.

STEP 1 Begin with a new, blank C++ file and enter the standard headers:

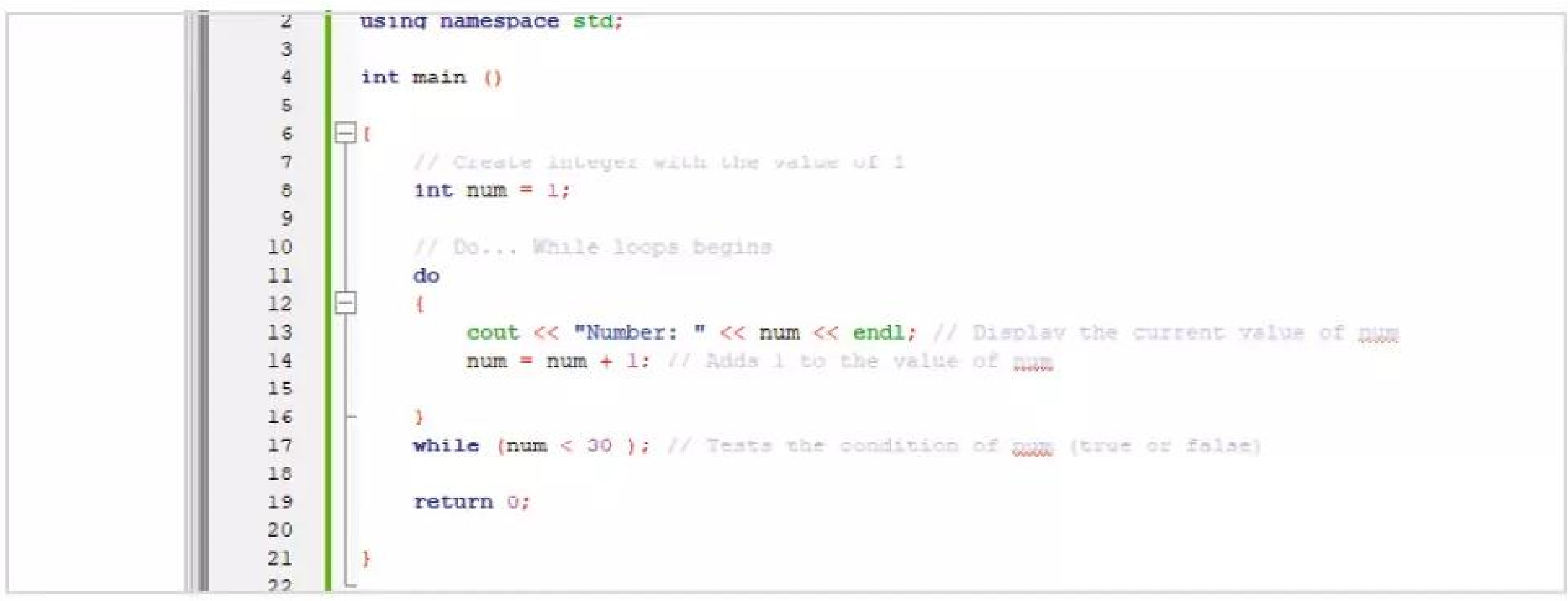
```
#include <iostream>
using namespace std;

int main ()

{


}
```



```
#include <iostream>
using namespace std;

int main ()

{


}
```

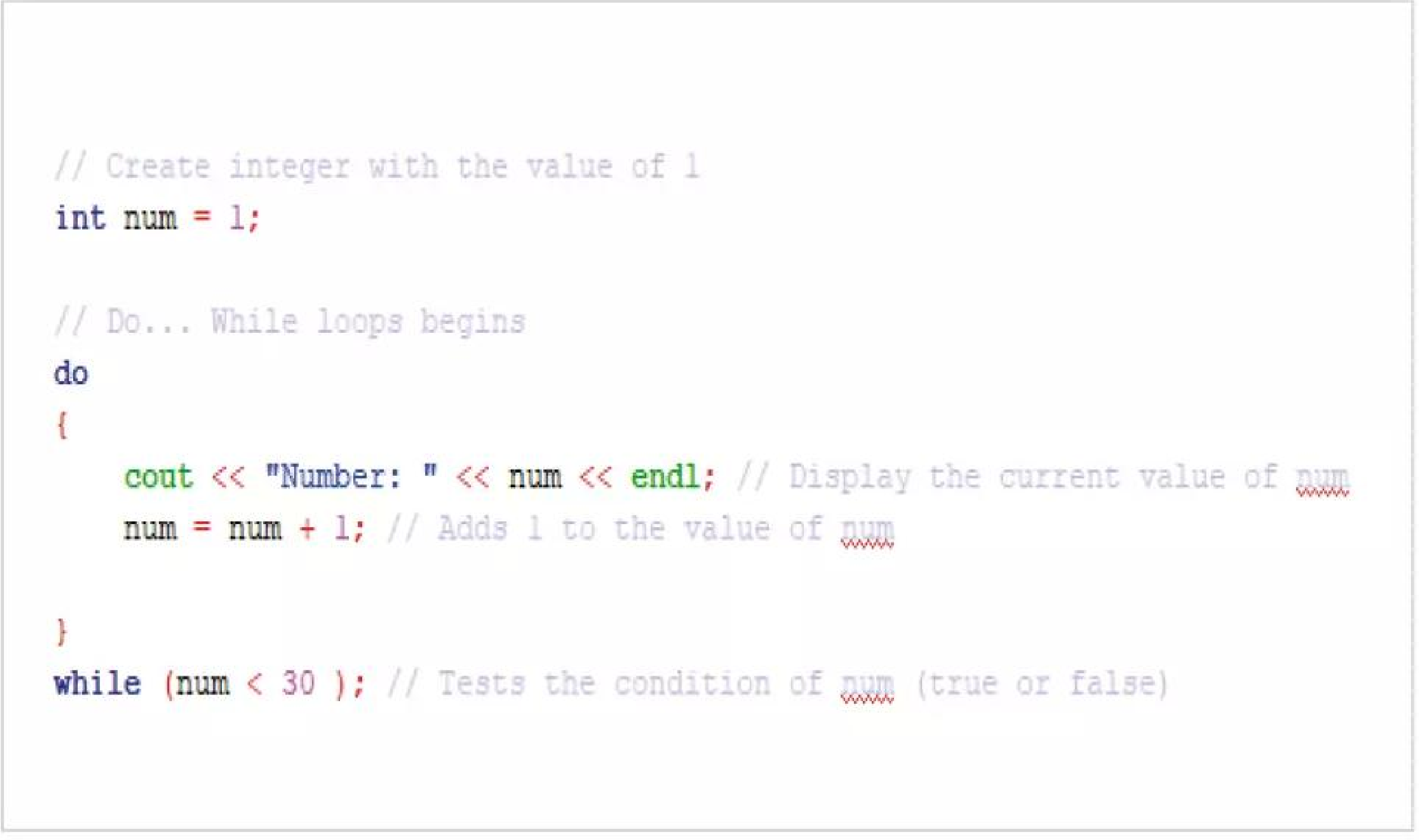
STEP 2 Begin with a simple number count:

```
{
    int num = 1;

    do
    {
        cout << “Number: “ << num << endl;
        num = num + 1;

    }
    while (num < 30 );

    return 0;

}
```



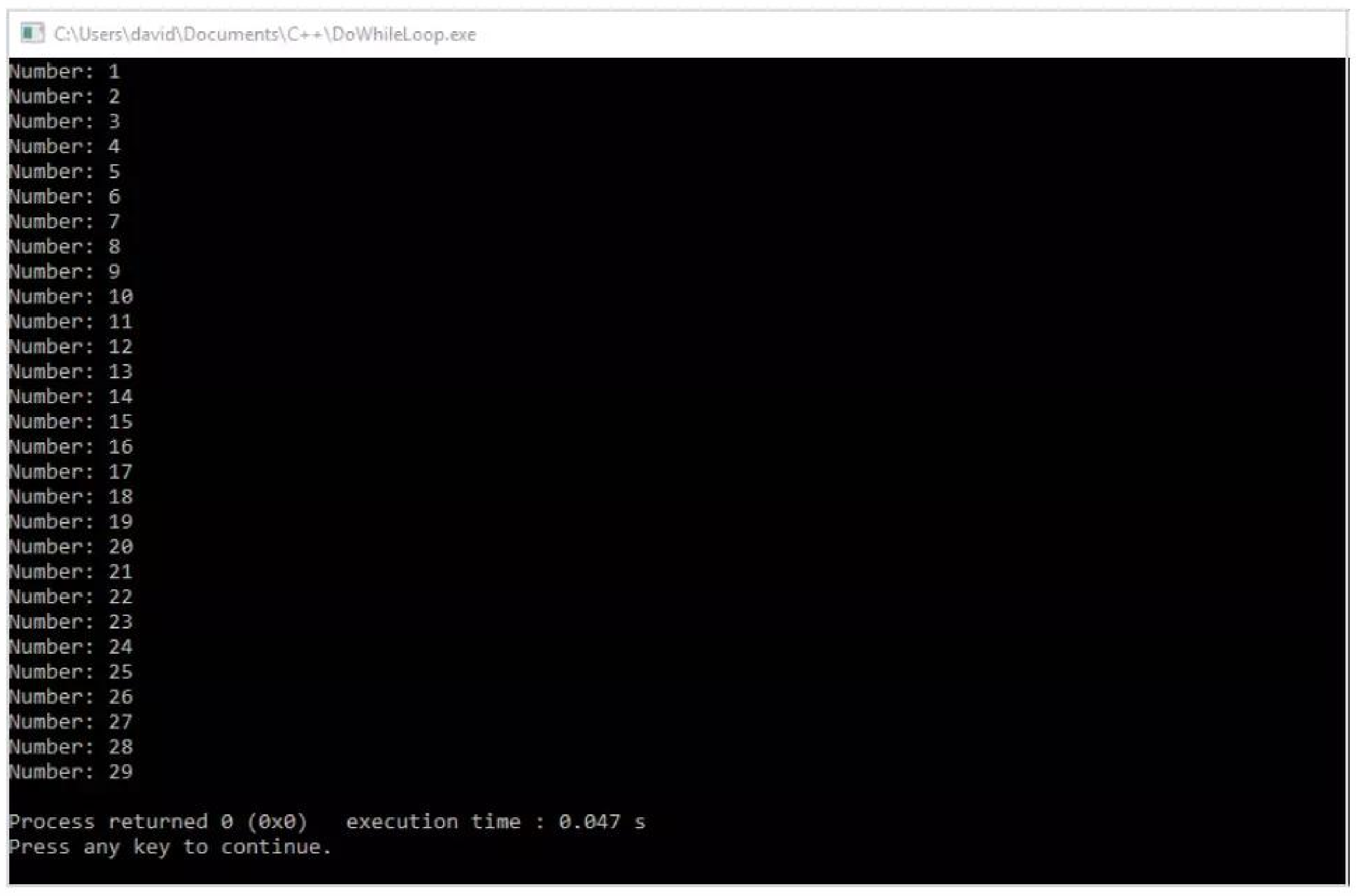
```
using namespace std;

int main ()

{
    // Create integer with the value of 1
    int num = 1;

    // Do... While loops begins
    do
    {
        cout << "Number: " << num << endl; // Display the current value of num
        num = num + 1; // Adds 1 to the value of num

    }
    while (num < 30 ); // Tests the condition of num (true or false)

    return 0;

}
```

STEP 3 Now, here's a look at the structure of a do… while loop. First you create an integer called num, with the value of 1. Now the do… while loops begins. The code inside the body of the loop is executed at least once, then the condition is checked for either true or false.



```
// Create integer with the value of 1
int num = 1;

// Do... While loops begins
do
{
    cout << "Number: " << num << endl; // Display the current value of num
    num = num + 1; // Adds 1 to the value of num

}
while (num < 30 ); // Tests the condition of num (true or false)
```

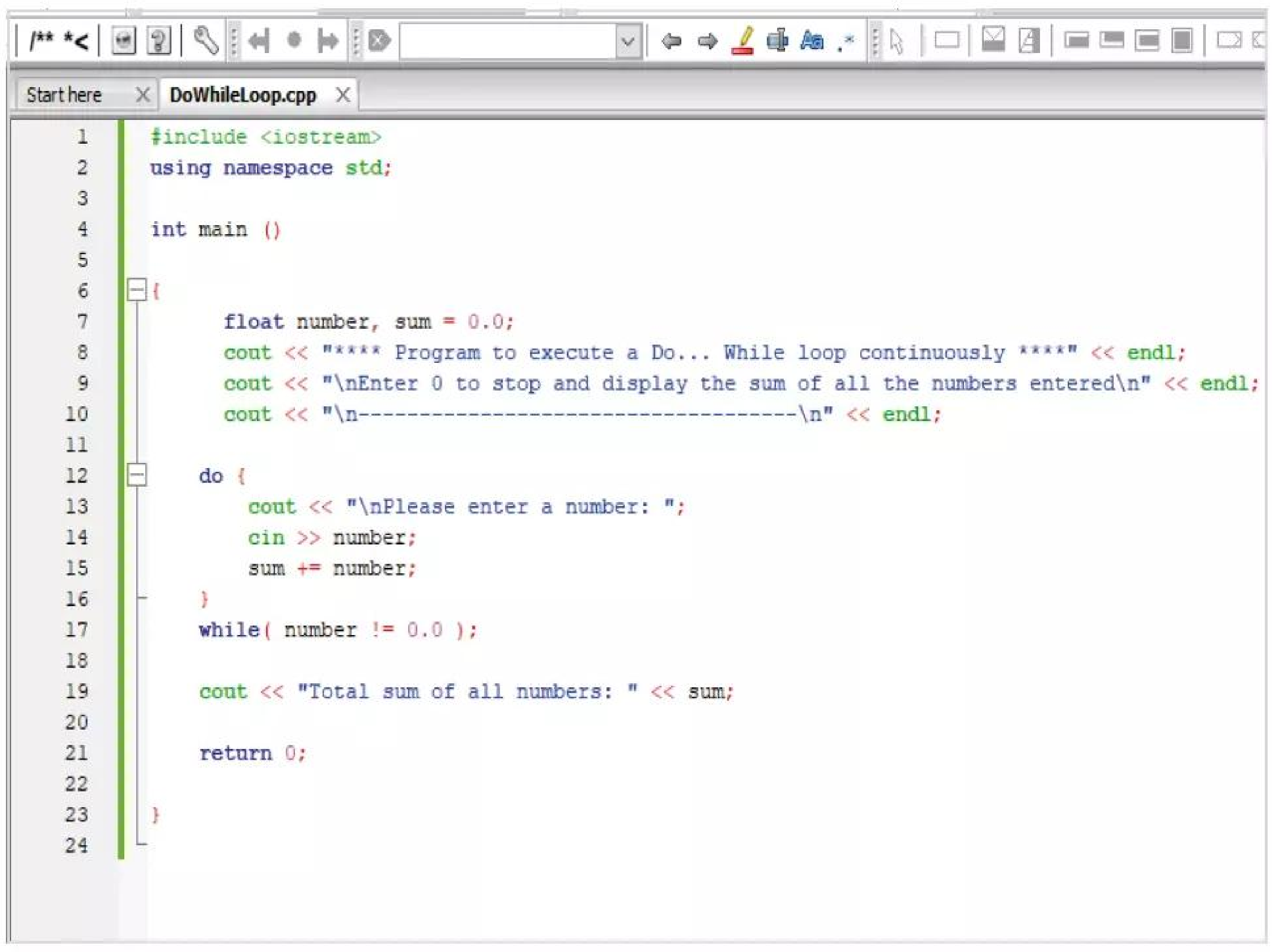
STEP 4 If the condition is true, the loop is executed. This continues until the condition is false. When the condition has been expressed as false, the loop terminates and the code continues. This means you can create a loop where the code continues until the user enters a certain character.



```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Number: 11
Number: 12
Number: 13
Number: 14
Number: 15
Number: 16
Number: 17
Number: 18
Number: 19
Number: 20
Number: 21
Number: 22
Number: 23
Number: 24
Number: 25
Number: 26
Number: 27
Number: 28
Number: 29

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

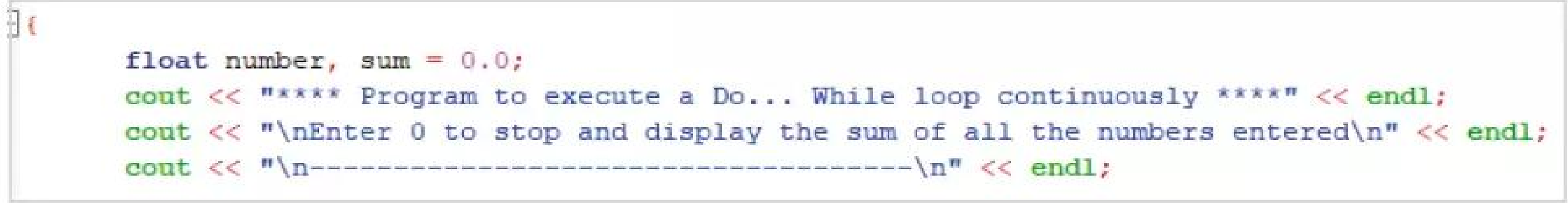
STEP 5 If you want code to add up user inputted numbers until the user enters zero:

```
{
      float number, sum = 0.0;
      cout << “**** Program to execute a Do... While loop continuously ****” << endl;
      cout << “\nEnter 0 to stop and display the sum of all the numbers entered\n” << endl;
      cout << “\n-------------------------------------\n” << endl;

    do {
        cout<<”\nPlease enter a number: “;
        cin>>number;
        sum += number;
    }
    while(number != 0.0);

    cout<<”Total sum of all numbers: “<<sum;

    return 0;
}
```



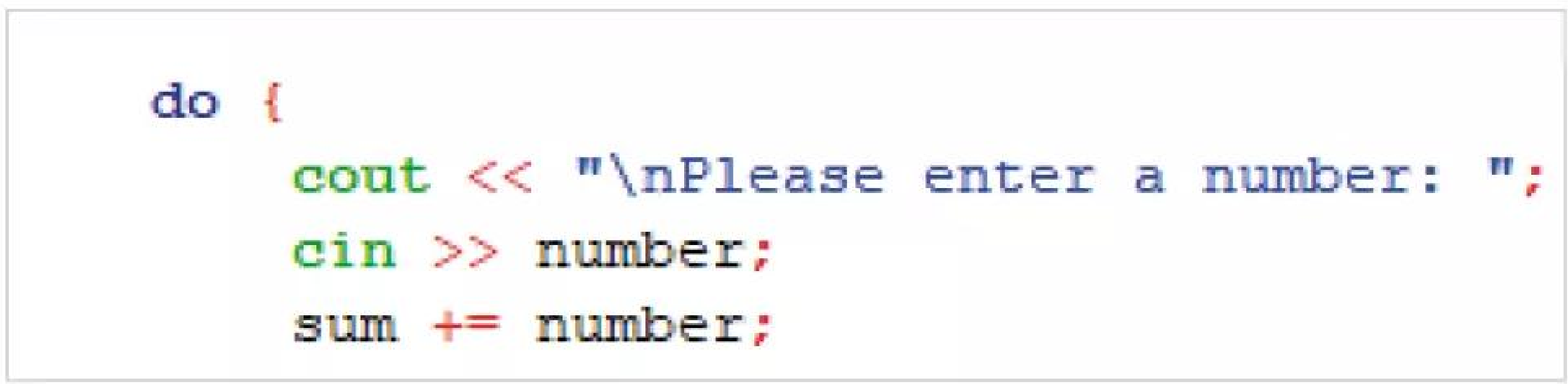
```
#include <iostream>
using namespace std;

int main ()

{

      float number, sum = 0.0;
      cout << "**** Program to execute a Do... While loop continuously ****" << endl;
      cout << "\nEnter 0 to stop and display the sum of all the numbers entered\n" << endl;
      cout << "\n-----------------------------------\n" << endl;

    do {
        cout << "\nPlease enter a number: ";
        cin >> number;
        sum += number;
    }
    while( number != 0.0 );

    cout << "Total sum of all numbers: " << sum;

    return 0;

}
```

STEP 6 The code from Step 5 works as follows: two floating point variables are assigned, number and sum, both with the value of 0.0. There is a brief set of instructions for the user, then the do… while loop begins.



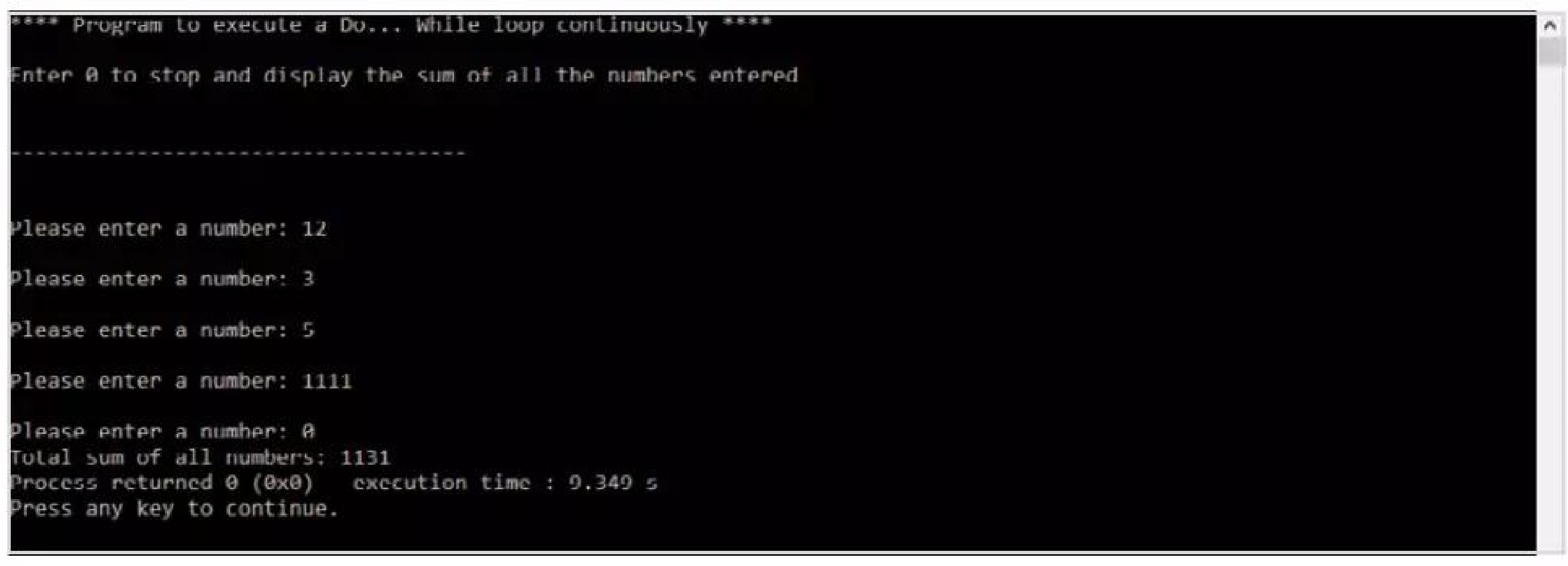
```
{
      float number, sum = 0.0;
      cout << "**** Program to execute a Do... While loop continuously ****" << endl;
      cout << "\nEnter 0 to stop and display the sum of all the numbers entered\n" << endl;
      cout << "\n-----------------------------------\n" << endl;
```

STEP 7 The do… while loop in this instance asks the user to input a number, which you assigned to the float variable, number. The calculation step uses the second floating point variable, sum, which adds the value of number every time the user enters a new value.



```
do {
    cout << "\nPlease enter a number: ";
    cin >> number;
    sum += number;
```

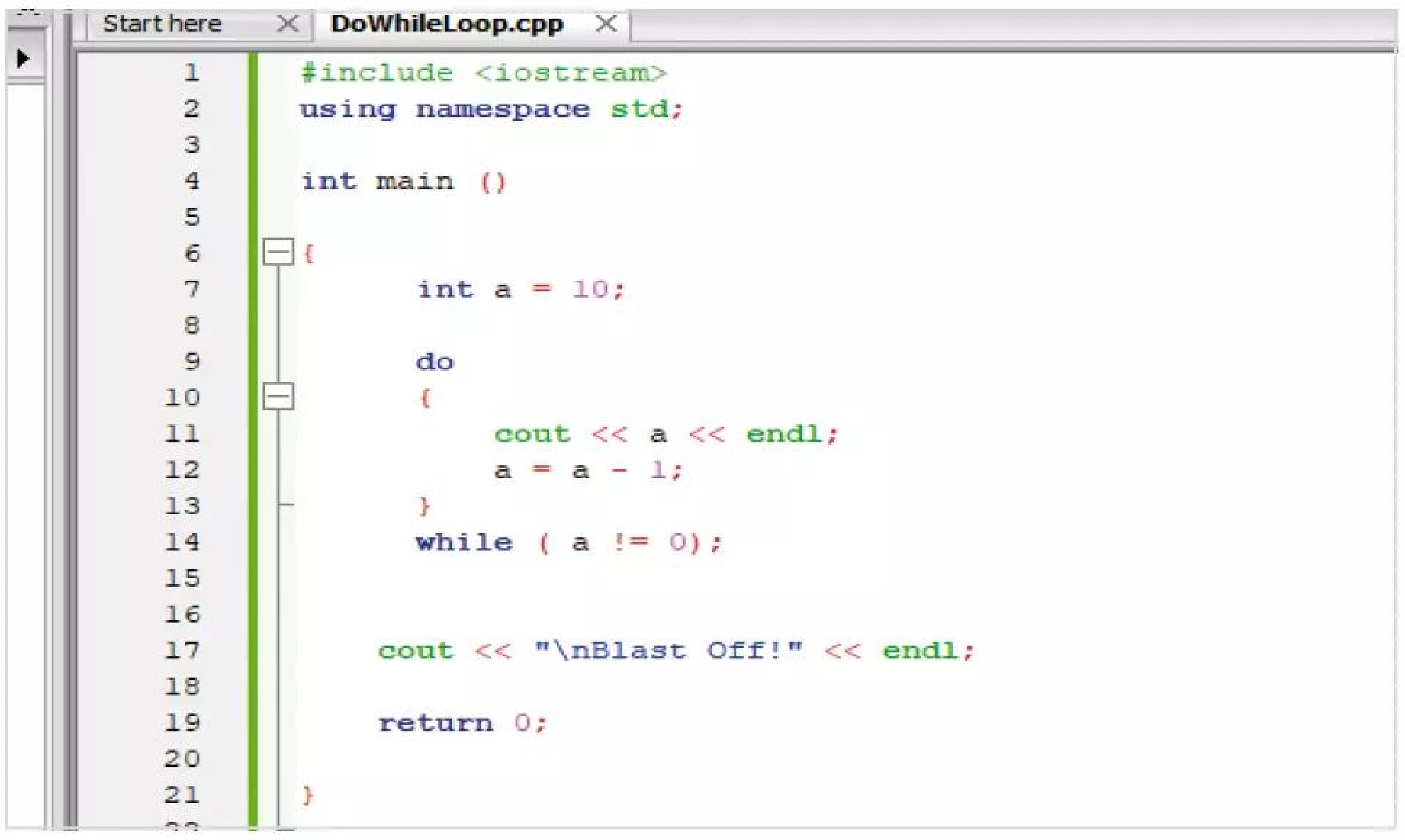
STEP 8 Finally, the while statement checks the condition of the variable number. If the user has entered zero, then the loop is terminated, if not then it continues indefinitely. When the user finally enters zero, the value of sum, the total value of all the user's input, is displayed. The loop, and the program, then ends.



```
**** Program to execute a Do... While loop continuously ****

Enter 0 to stop and display the sum of all the numbers entered


-----------------------------------


Please enter a number: 12

Please enter a number: 3

Please enter a number: 5

Please enter a number: 1111

Please enter a number: 0
Total sum of all numbers: 1131
Process returned 0 (0x0)   execution time : 9.349 s
Press any key to continue.
```

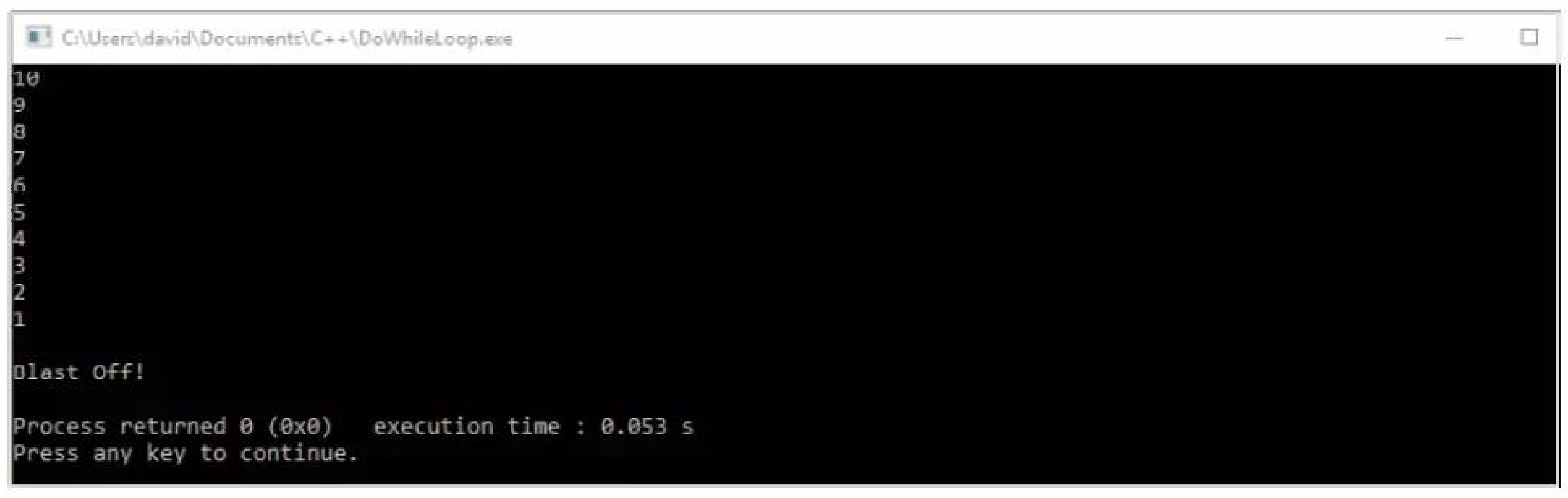
STEP 9 Using the countdown and Blast Off! code used previously, a do… while loop would look like:

```
{
      int a = 10;

      do
      {
          cout << a << endl;
          a = a - 1;
      }
      while ( a != 0);


    cout << “\nBlast Off!” << endl;

    return 0;
}
```



```
#include <iostream>
using namespace std;

int main ()

{

      int a = 10;

      do
      {
          cout << a << endl;
          a = a - 1;
      }
      while ( a != 0);


    cout << "\nBlast Off!" << endl;

    return 0;

}
```

STEP 10 The main advantage of using a do… while loop is because it's an exit-condition loop; whereas a while loop is an entry-control loop. Therefore, if your code requires a loop that needs to be executed at least once (for example, to check the number of lives in a game), then a do… while loop is perfect.



```
10
9
8
7
6
5
4
3
2
1

Blast Off!

Process returned 0 (0x0)   execution time : 0.053 s
Press any key to continue.
```

If Statement

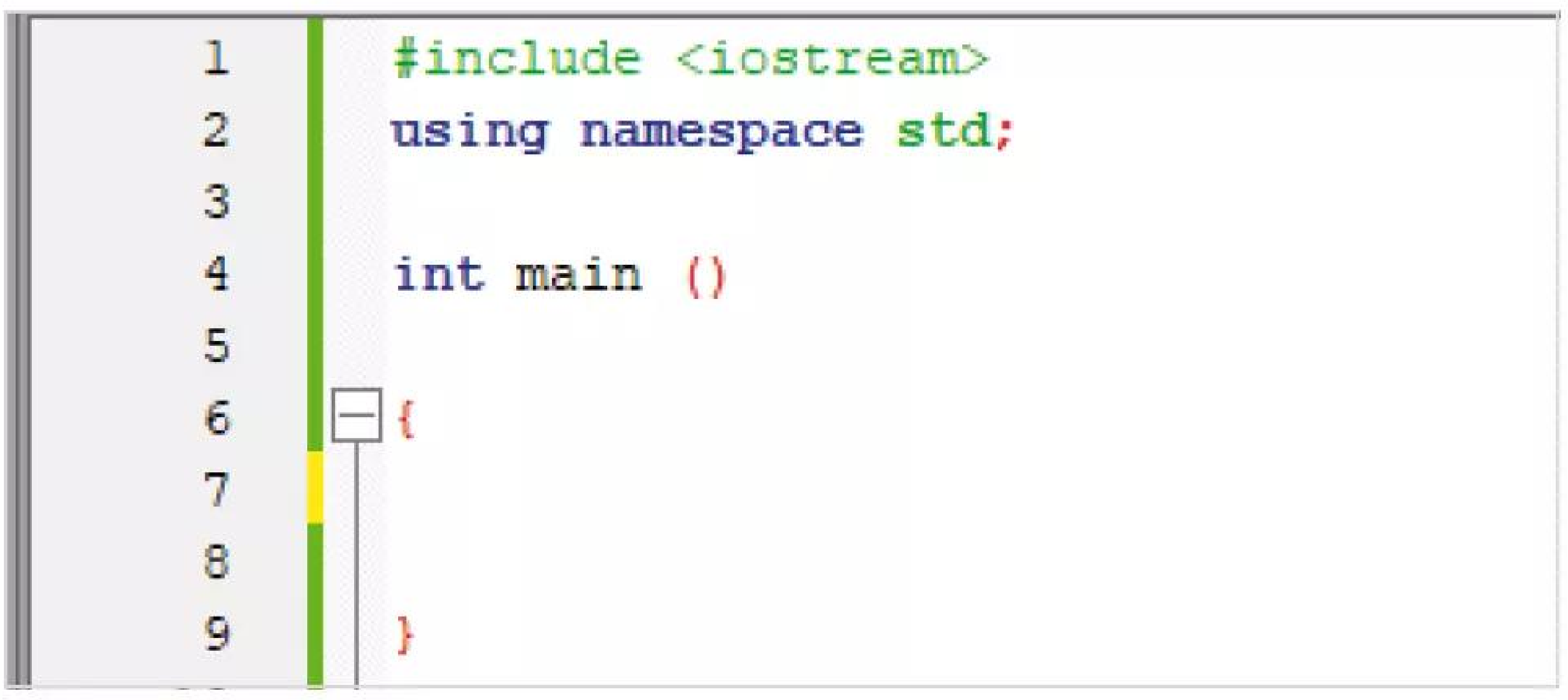
The decision making statement ‘if’ is probably one of the most used statements in any programming language, regardless of whether it’s C++, Python, BASIC or anything else. It represents a junction in the code, where IF one condition is true, do this; or IF it’s false, do that.

IF ONLY

If uses a Boolean expression within its statement. If the Boolean expression is true, the code within the statement is executed. If not, then the code after the statement is executed instead.

STEP 1 First, create a new C++ file and enter the relevant standard headers, as usual:

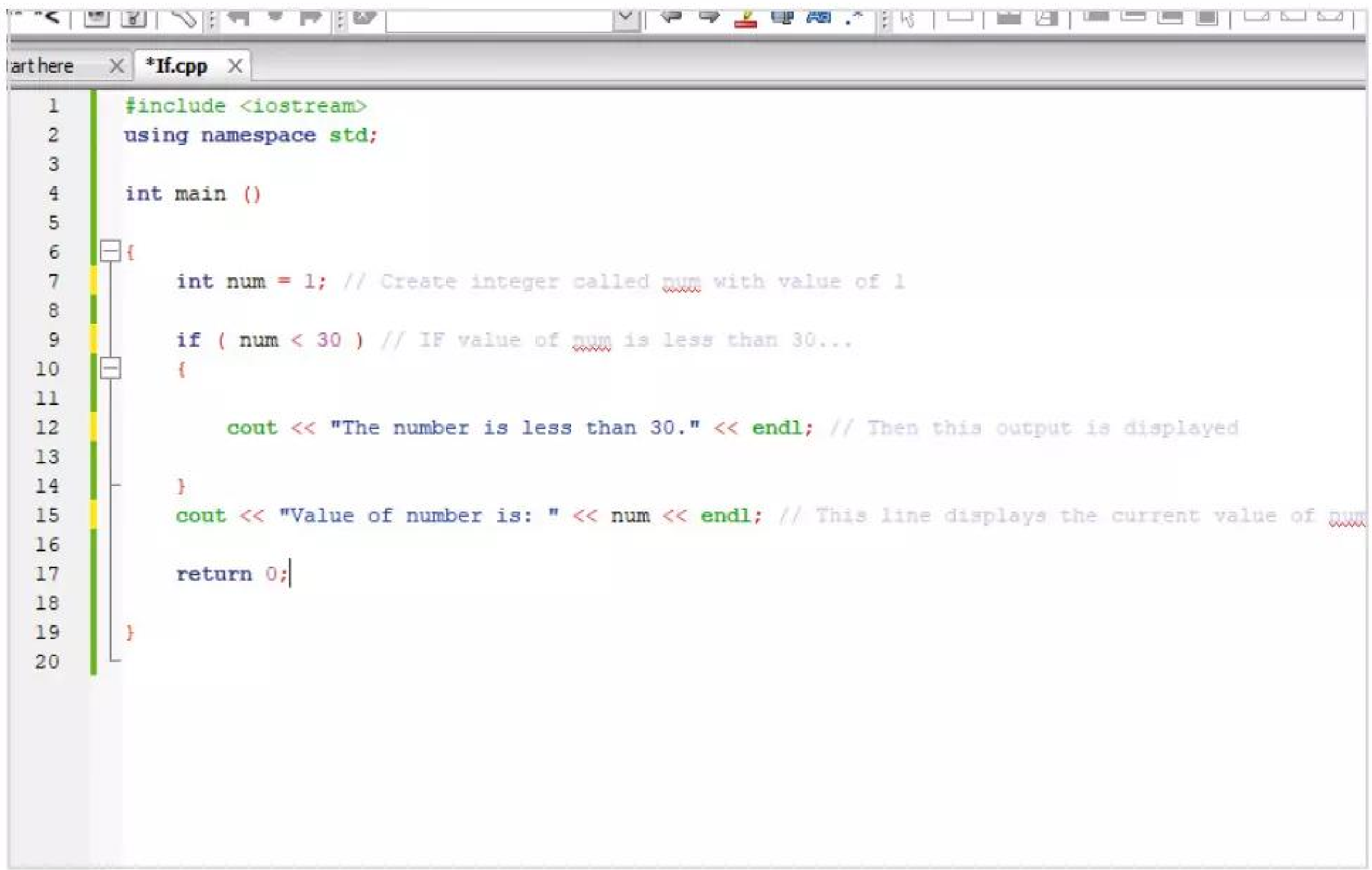
```
#include <iostream>
using namespace std;

int main ()

{


}
```



```
#include <iostream>
using namespace std;

int main ()

{


}
```

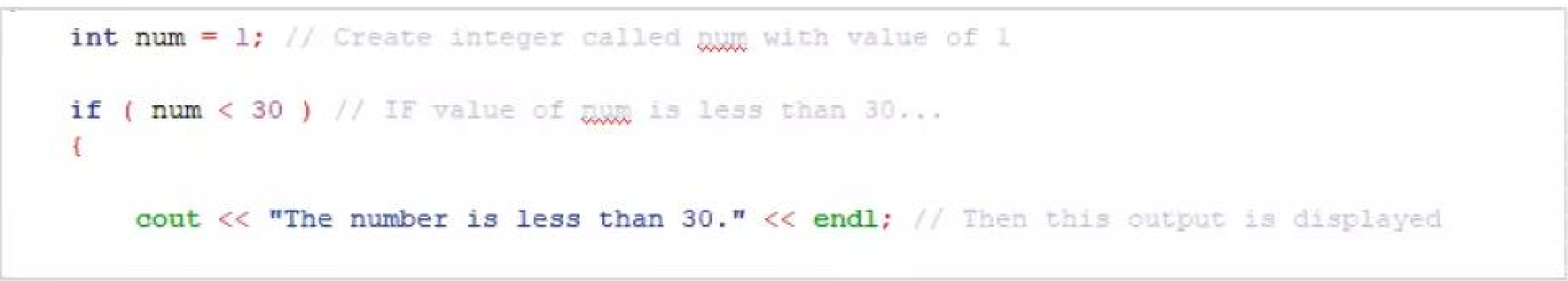
STEP 2 If is best explained when you use a number-based condition:

```
{
    int num = 1;

    if ( num < 30 )
    {

        cout << “The number is less than 30.” <<
endl;

    }
    cout << “Value of number is: “ << num << endl;

    return 0;

}
```



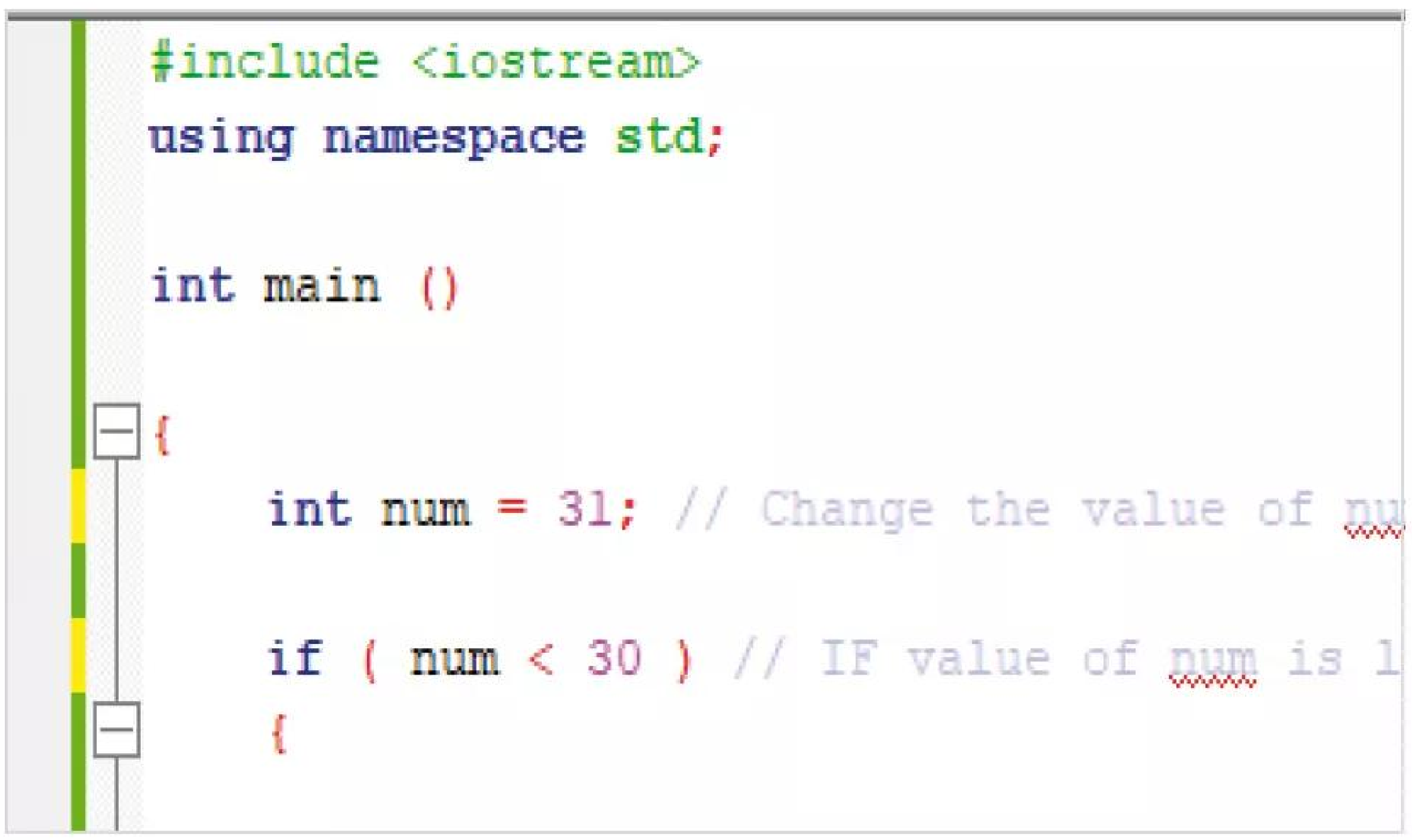
```
#include <iostream>
using namespace std;

int main ()

{
    int num = 1; // Create integer called num with value of 1

    if ( num < 30 ) // IF value of num is less than 30...
    {

        cout << "The number is less than 30." << endl; // Then this output is displayed

    }
    cout << "Value of number is: " << num << endl; // This line displays the current value of num

    return 0;

}
```

STEP 3 What’s going on here? To begin, an integer called num was created and assigned with the value of 1. The if statement comes next, and in this case we’ve instructed the code that if the condition, the value, of num is less than 1, then the code within the braces should be executed.



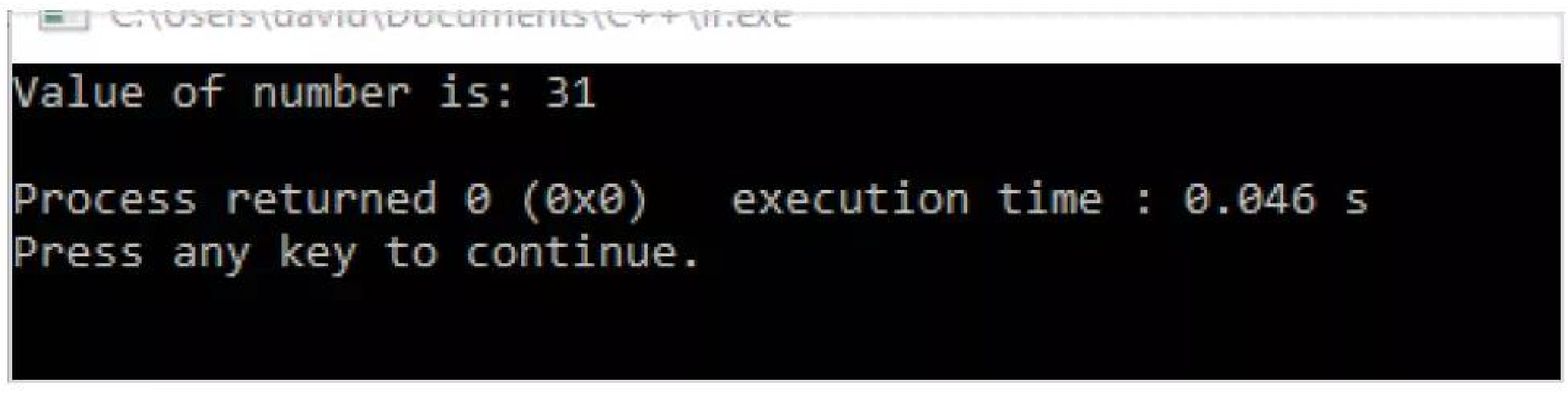
```
int num = 1; // Create integer called num with value of 1

if ( num < 30 ) // IF value of num is less than 30...
{

    cout << "The number is less than 30." << endl; // Then this output is displayed
```

STEP 4 The second cout statement displays the current value of num and the program terminates safely. It’s easy to see how the if statement works if you were to change the initial value of num from 1 to 31.



```
#include <iostream>
using namespace std;

int main ()

{

    int num = 31; // Change the value of nu

    if ( num < 30 ) // IF value of num is l
    {
```

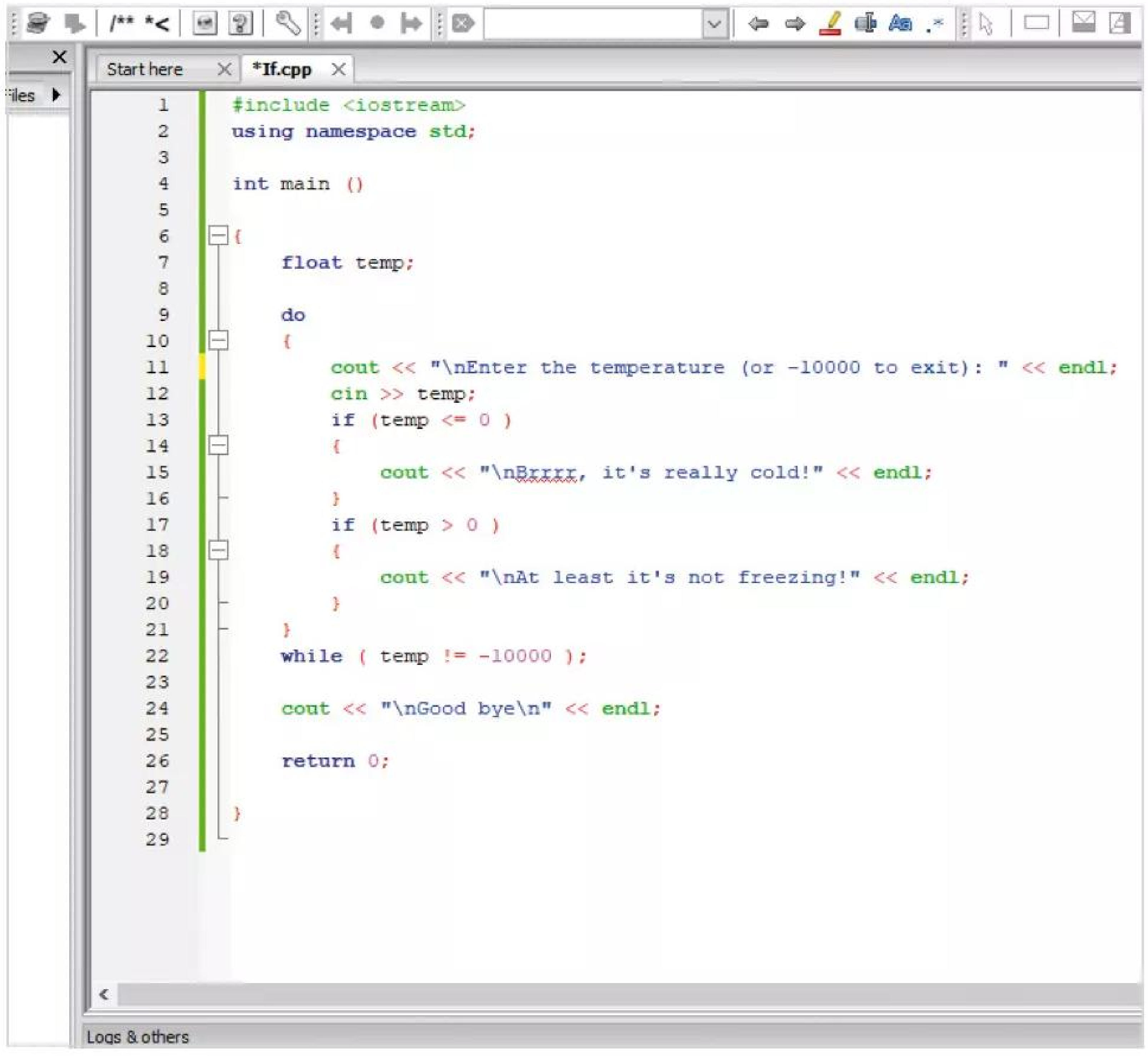
STEP 5 When you change the value to anything above 30, then build and run the code, you can see that the only line to be outputted to the screen is the second cout statement, displaying the current value of num. This is because the initial if statement is false, so it ignores the code within the braces.



```
Value of number is: 31

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

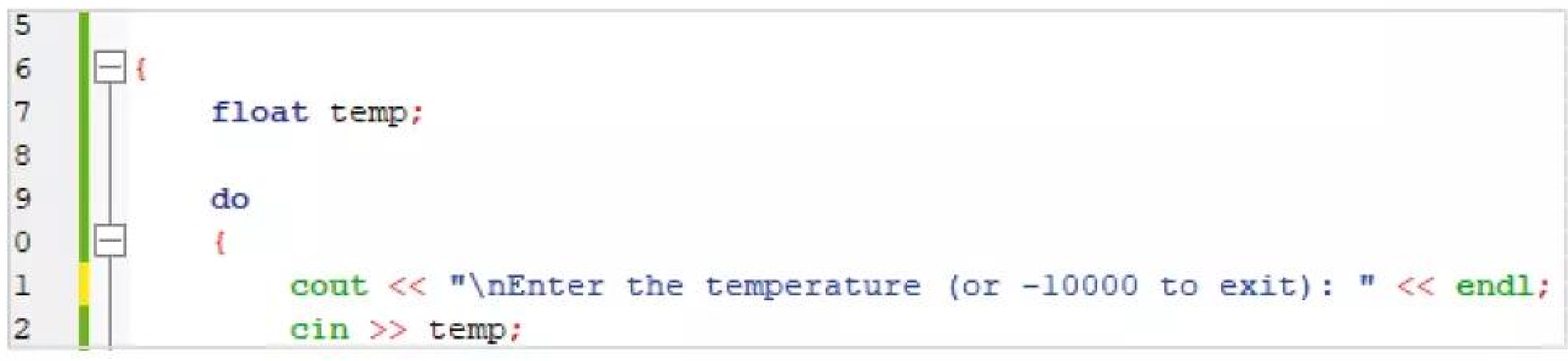
STEP 6 You can include an if statement within a do… while loop. For example:

```
{
    float temp;

    do
    {
        cout << “\nEnter the temperature (or
-10000 to exit): “ << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << “\nBrrrr, it’s really cold!”
<< endl;
        }
        if (temp > 0 )
        {
            cout << “\nAt least it’s not
freezing!” << endl;
        }
    }
    while ( temp != -10000 );

    cout << “\nGood bye\n” << endl;

    return 0;
}
```



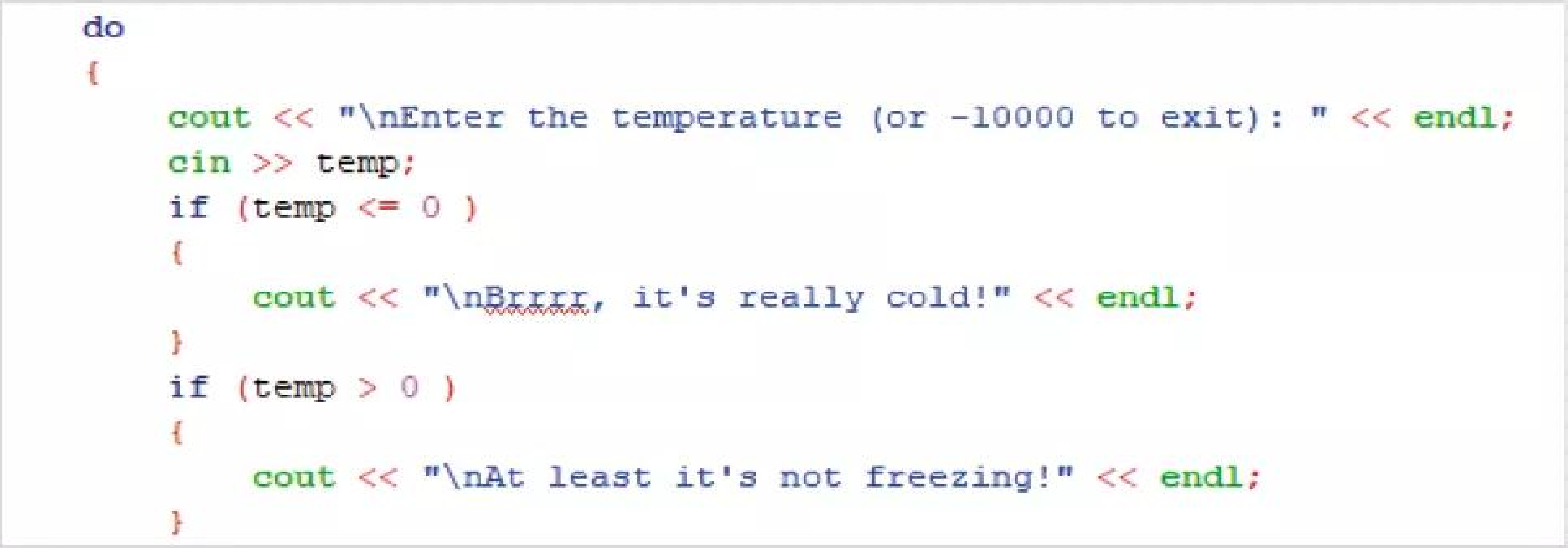
```
#include <iostream>
using namespace std;

int main ()

{

    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << "\nBrrrr, it's really cold!" << endl;
        }
        if (temp > 0 )
        {
            cout << "\nAt least it's not freezing!" << endl;
        }
    }
    while ( temp != -10000 );

    cout << "\nGood bye\n" << endl;

    return 0;
}
```

STEP 7 The code in Step 6 is simplistic but effective. First we created a floating point integer called temp, then a do… while loop that asks the user to enter the current temperature.



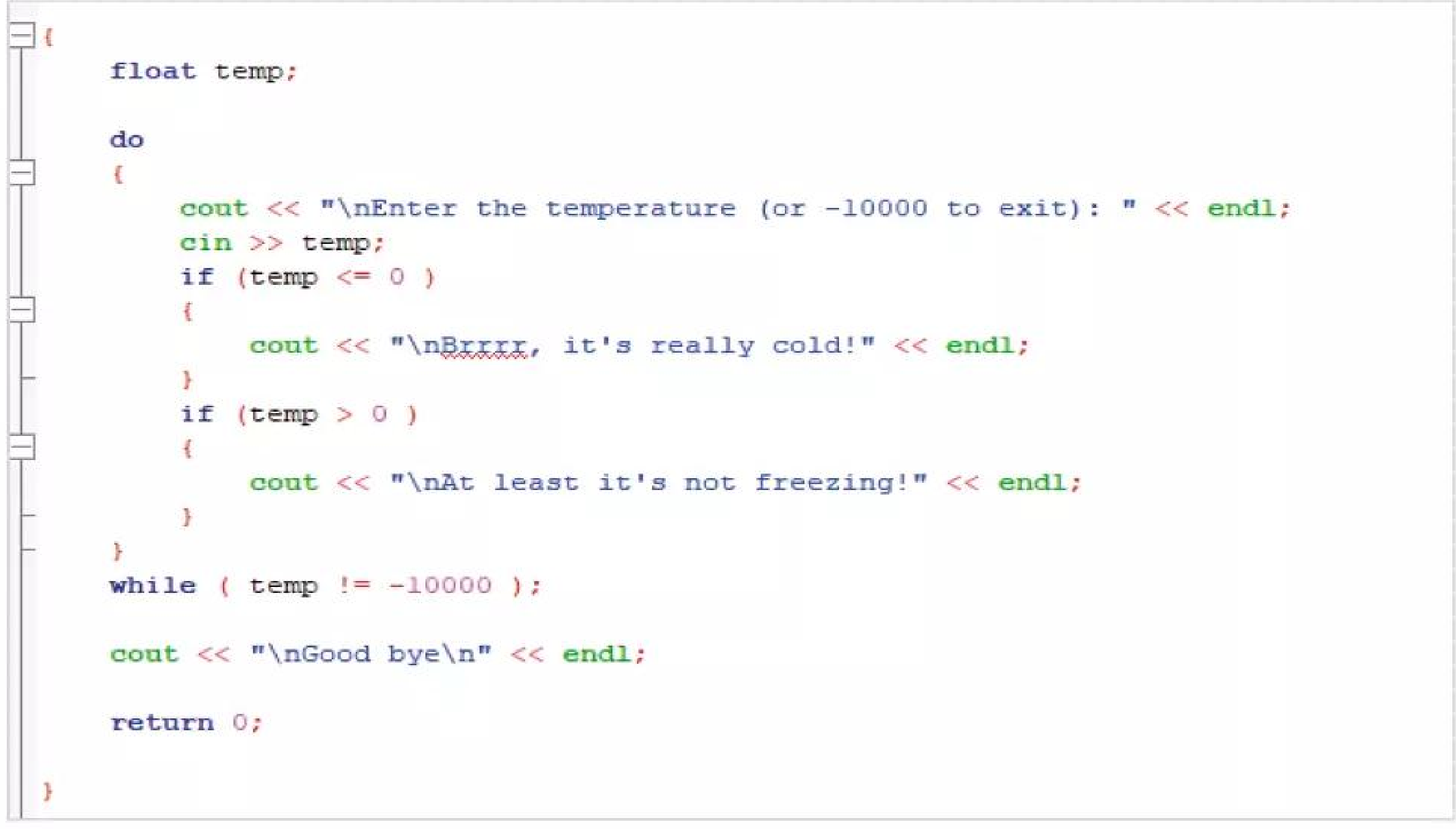
```
{
    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
```

STEP 8 The first if statement checks to see if the user’s inputted value is less than or equal to zero. If it is, then the output is ‘Brrrr, it’s really cold!’. Otherwise, if the input is greater than zero, the code outputs ‘At least it’s not freezing!’.



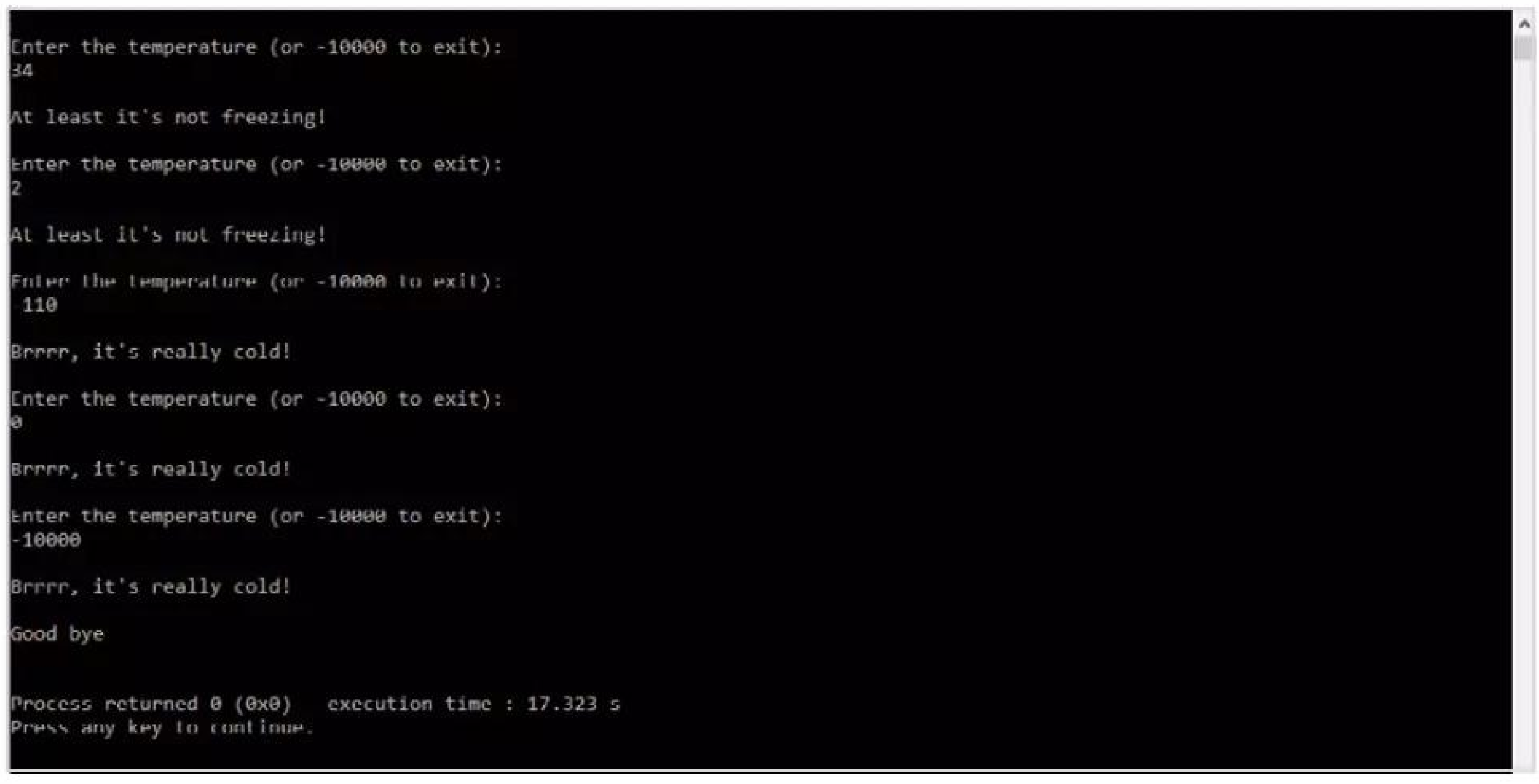
```
do
{
    cout << "\nEnter the temperature (or -10000 to exit): " << endl;
    cin >> temp;
    if (temp <= 0 )
    {
        cout << "\nBrrrr, it's really cold!" << endl;
    }
    if (temp > 0 )
    {
        cout << "\nAt least it's not freezing!" << endl;
    }
```

STEP 9 Finally, if the user enters the value -10000, which is impossibly cold so is therefore a unrealistic value, the do… while loop is terminated and a friendly ‘Good bye’ is displayed to the screen.



```
{
    float temp;

    do
    {
        cout << "\nEnter the temperature (or -10000 to exit): " << endl;
        cin >> temp;
        if (temp <= 0 )
        {
            cout << "\nBrrrr, it's really cold!" << endl;
        }
        if (temp > 0 )
        {
            cout << "\nAt least it's not freezing!" << endl;
        }
    }
    while ( temp != -10000 );

    cout << "\nGood bye\n" << endl;

    return 0;
}
```

STEP 10 Using if is quite powerful, if it’s used correctly. Just remember that if the condition is true then the code executes what’s in the braces. If not, it continues on its merry way. See what else you can come up with using if and a combination of loops.



```
Enter the temperature (or -10000 to exit):
34

At least it's not freezing!

Enter the temperature (or -10000 to exit):
2

At least it's not freezing!

Enter the temperature (or -10000 to exit):
 110

Brrrr, it's really cold!

Enter the temperature (or -10000 to exit):
0

Brrrr, it's really cold!

Enter the temperature (or -10000 to exit):
-10000

Brrrr, it's really cold!

Good bye


Process returned 0 (0x0)   execution time : 17.323 s
Press any key to continue.
```

If… Else Statement

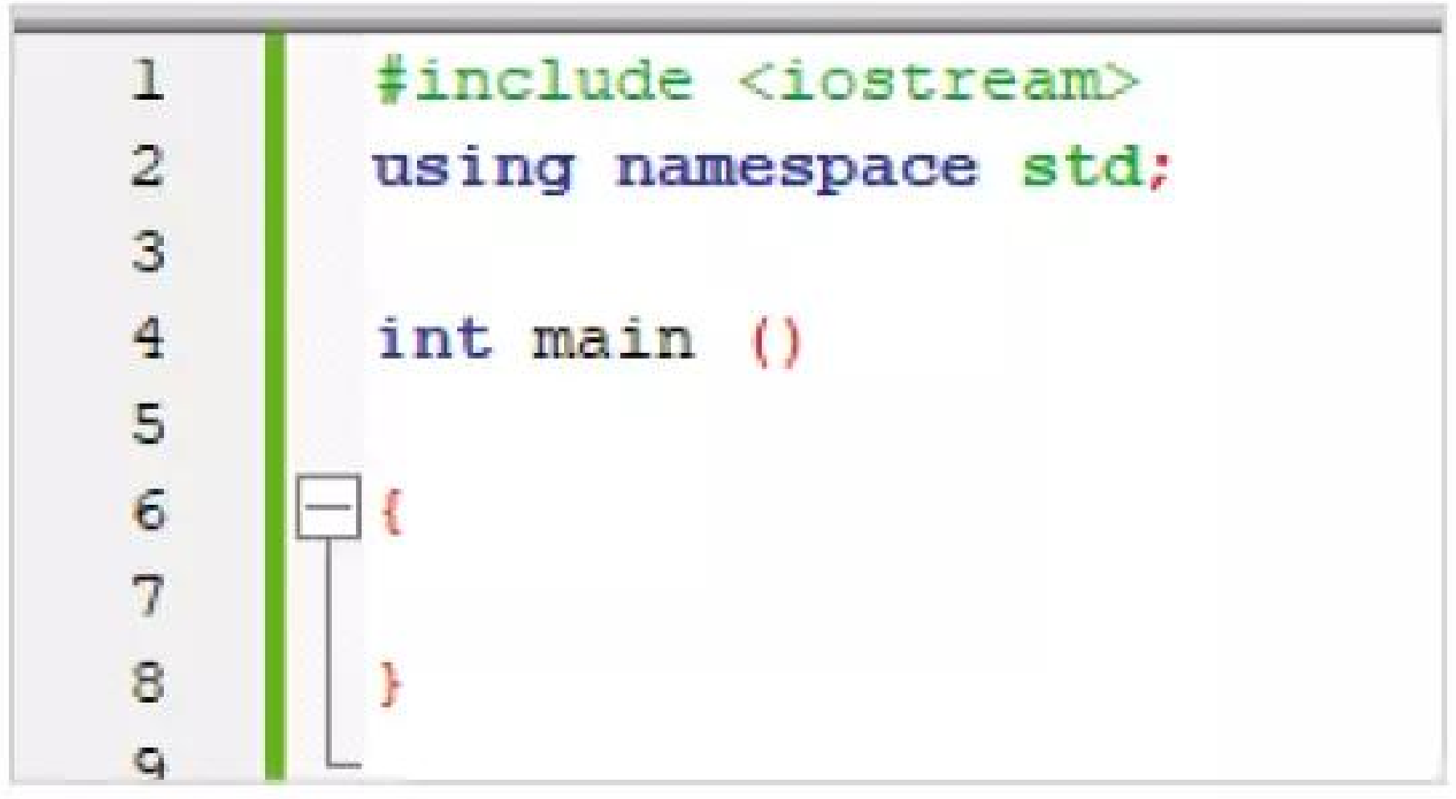
There is a much better way to use an if statement in your code, with if… else. If… else works in much the same way as a standard if statement. If the Boolean expression is true, the code with in the braces is executed. Else, the code within the next set of braces is used instead.

IF YES, ELSE NO

There are two sections of code that can be executed depending on the outcome in an if... else statement. It's quite easy to visualise once you get used to its structure.

STEP 1 Begin with a new C++ file and the standard headers:

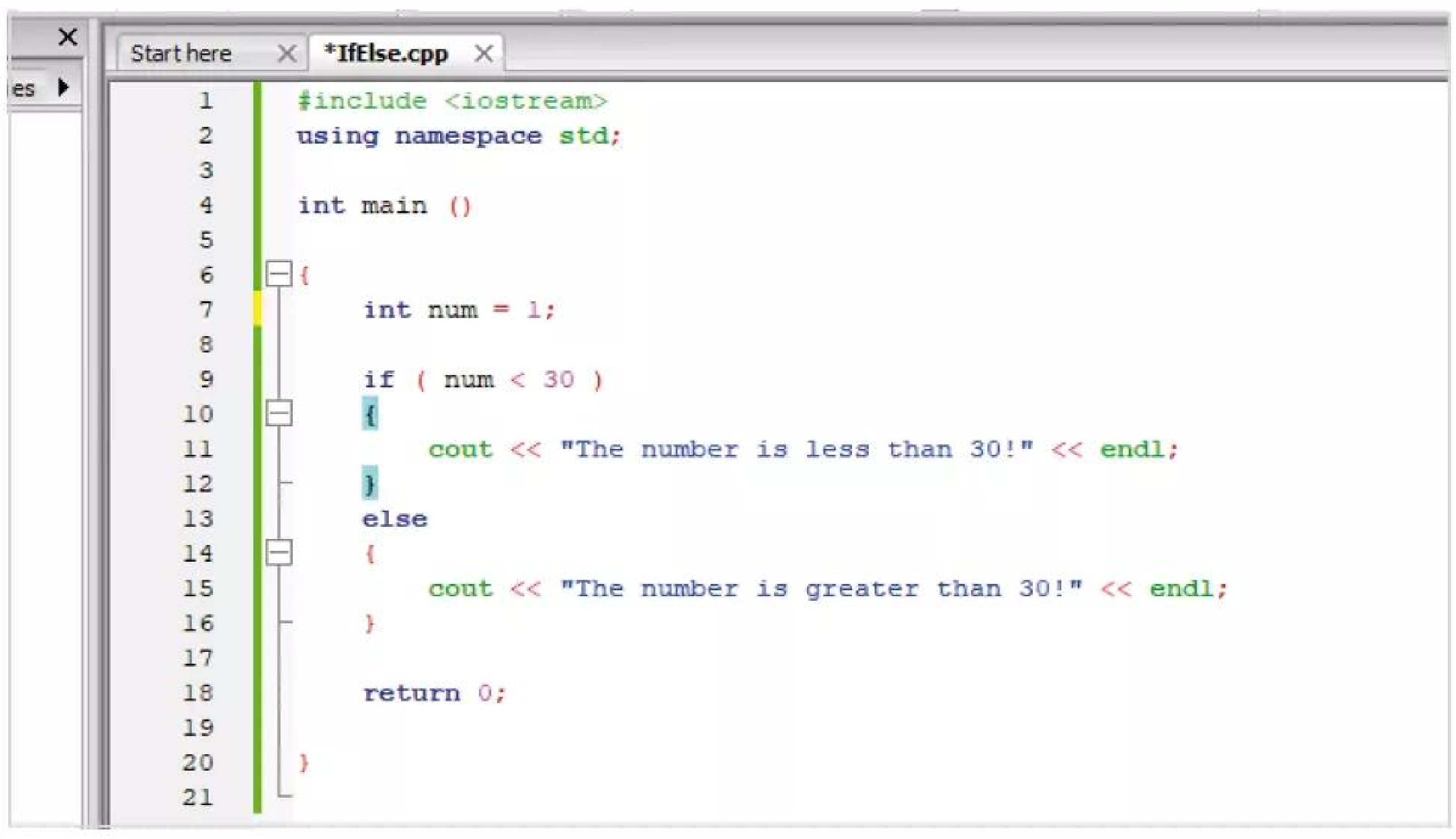
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{

}
```

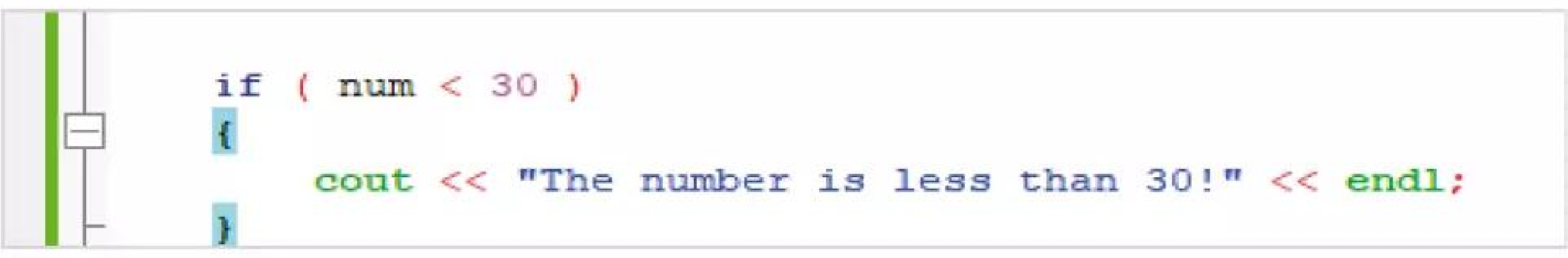
STEP 2 Let's expand the code from the If Statement on the previous page:

```
{
    int num = 1;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else
    {
        cout << “The number is greater than 30!”
<< endl;
    }

    return 0;

}
```



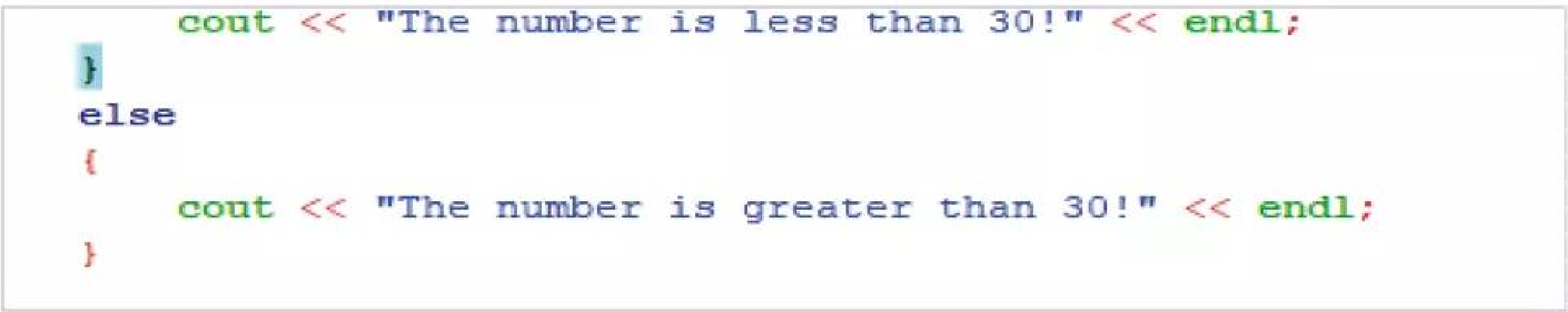
```
#include <iostream>
using namespace std;

int main ()

{
    int num = 1;

    if ( num < 30 )
    {
        cout << "The number is less than 30!" << endl;
    }
    else
    {
        cout << "The number is greater than 30!" << endl;
    }

    return 0;

}
```

STEP 3 The first line in the code creates the integer called num and gives it a value of 1. The if statement checks to see if the value of num is less than thirty and if it is it outputs "The number is less than 30!" to the console.



```
    if ( num < 30 )
    {
        cout << "The number is less than 30!" << endl;
    }
```

STEP 4 The else companion to if checks if the number is greater than 30 and if so, then displays "The number is greater than 30!" to the console; and finally, the code is terminated satisfactorily.



```
        cout << "The number is less than 30!" << endl;
    }
    else
    {
        cout << "The number is greater than 30!" << endl;
    }
```

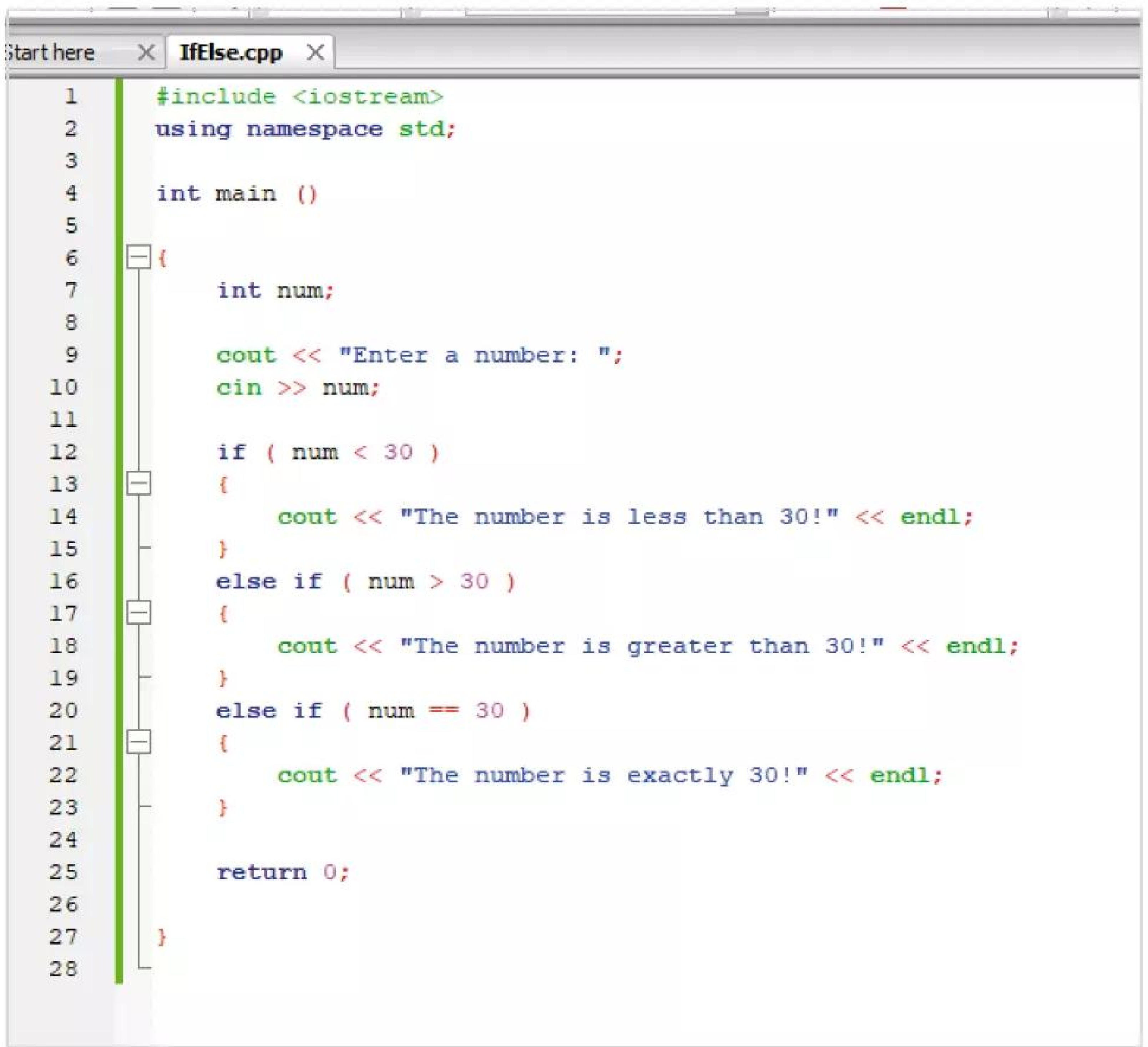
STEP 5 You can change the value of num in the code or you can improve the code by asking the user to enter a value:

```
{
    int num;

    cout << “Enter a number: “;
    cin >> num;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else
    {
        cout << “The number is greater than 30!”
<< endl;
    }

    return 0;

}
```

STEP 6

The code works the same way, as you would expect, but what if you wanted to display something if the user entered the number 30? Try this:

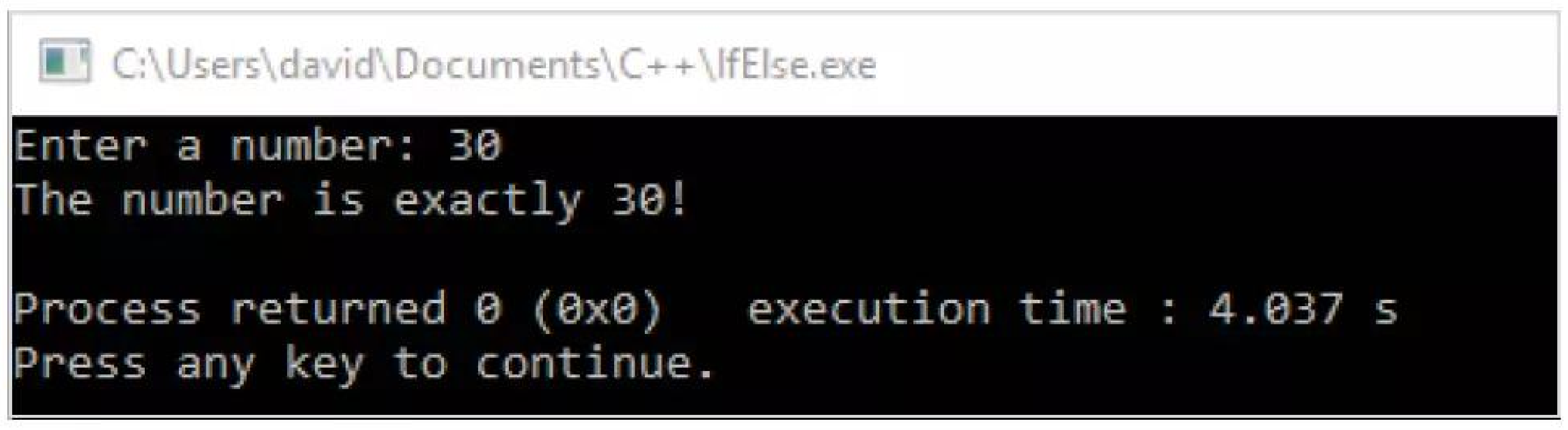
```
{
    int num;

    cout << “Enter a number: “;
    cin >> num;

    if ( num < 30 )
    {
        cout << “The number is less than 30!” <<
endl;
    }
    else if ( num > 30 )
    {
        cout << “The number is greater than 30!”
<< endl;
    }
    else if ( num == 30 )
    {
        cout << “The number is exactly 30!” <<
endl;
    }

    return 0;

}
```



```
#include <iostream>
using namespace std;

int main ()

{

    int num;

    cout << "Enter a number: ";
    cin >> num;

    if ( num < 30 )
    {
        cout << "The number is less than 30!" << endl;
    }
    else if ( num > 30 )
    {
        cout << "The number is greater than 30!" << endl;
    }
    else if ( num == 30 )
    {
        cout << "The number is exactly 30!" << endl;
    }

    return 0;

}
```

STEP 7

The new addition to the code is what's known as a nested if… else statement. This allows you to check for multiple conditions. In this case, if the user enters a number less than 30, greater than 30 or actually 30 itself, a different outcome is presented to them.



```
Enter a number: 30
The number is exactly 30!

Process returned 0 (0x0)   execution time : 4.037 s
Press any key to continue.
```

STEP 8

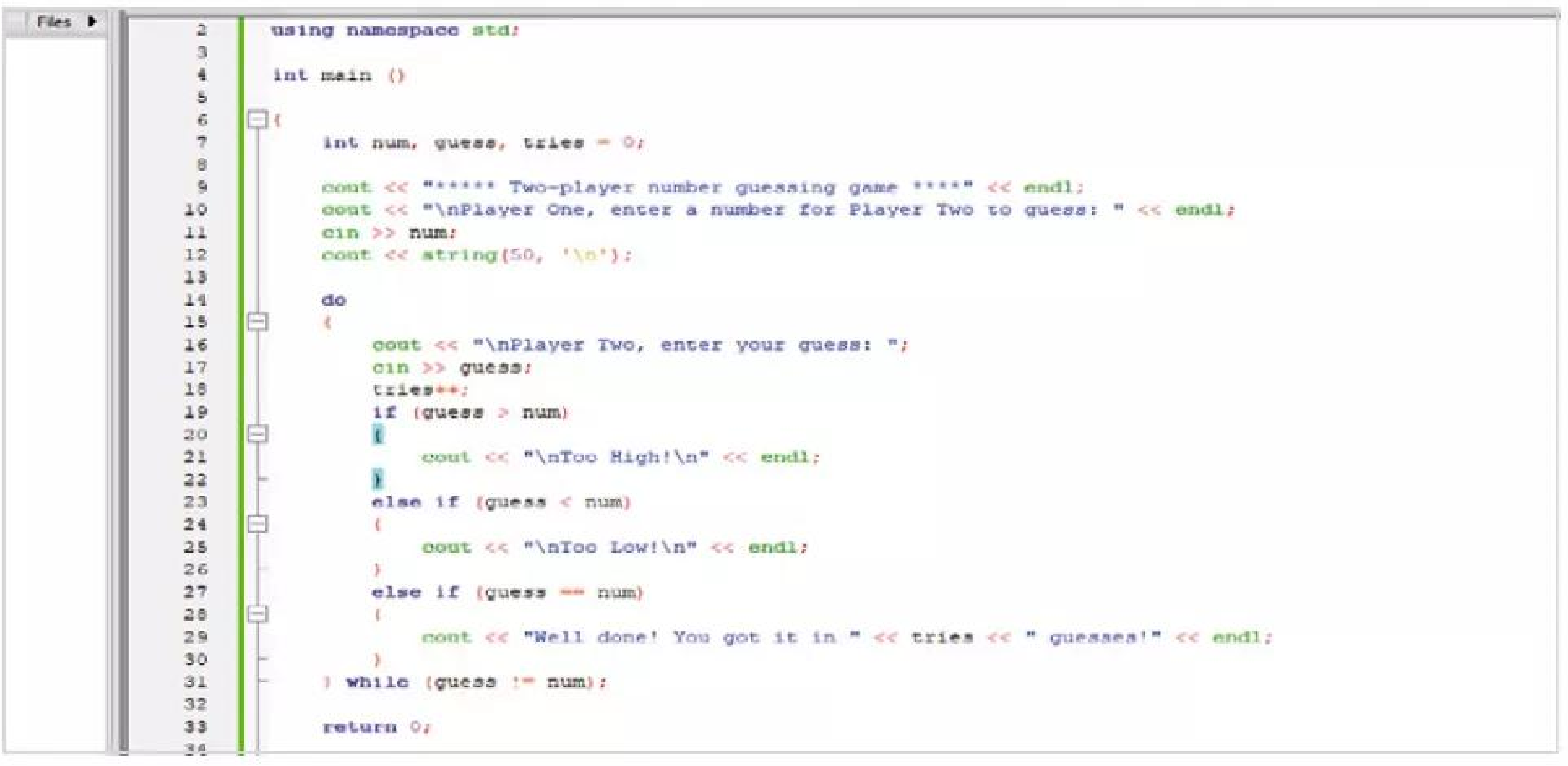
You can take this up a notch and create a two-player number guessing game. Begin by creating the variables:

```
int num, guess, tries = 0;
    cout << “***** Two-player number guessing game
****” << endl;
    cout << “\nPlayer One, enter a number for
Player Two to guess: “ << endl;
    cin >> num;
    cout << string(50, ‘\n’);
```

STEP 9

The cout << string(50, '\n') line clears the screen so Player Two doesn't see the entered number. Now you can create a do… while loop, together with if… else:

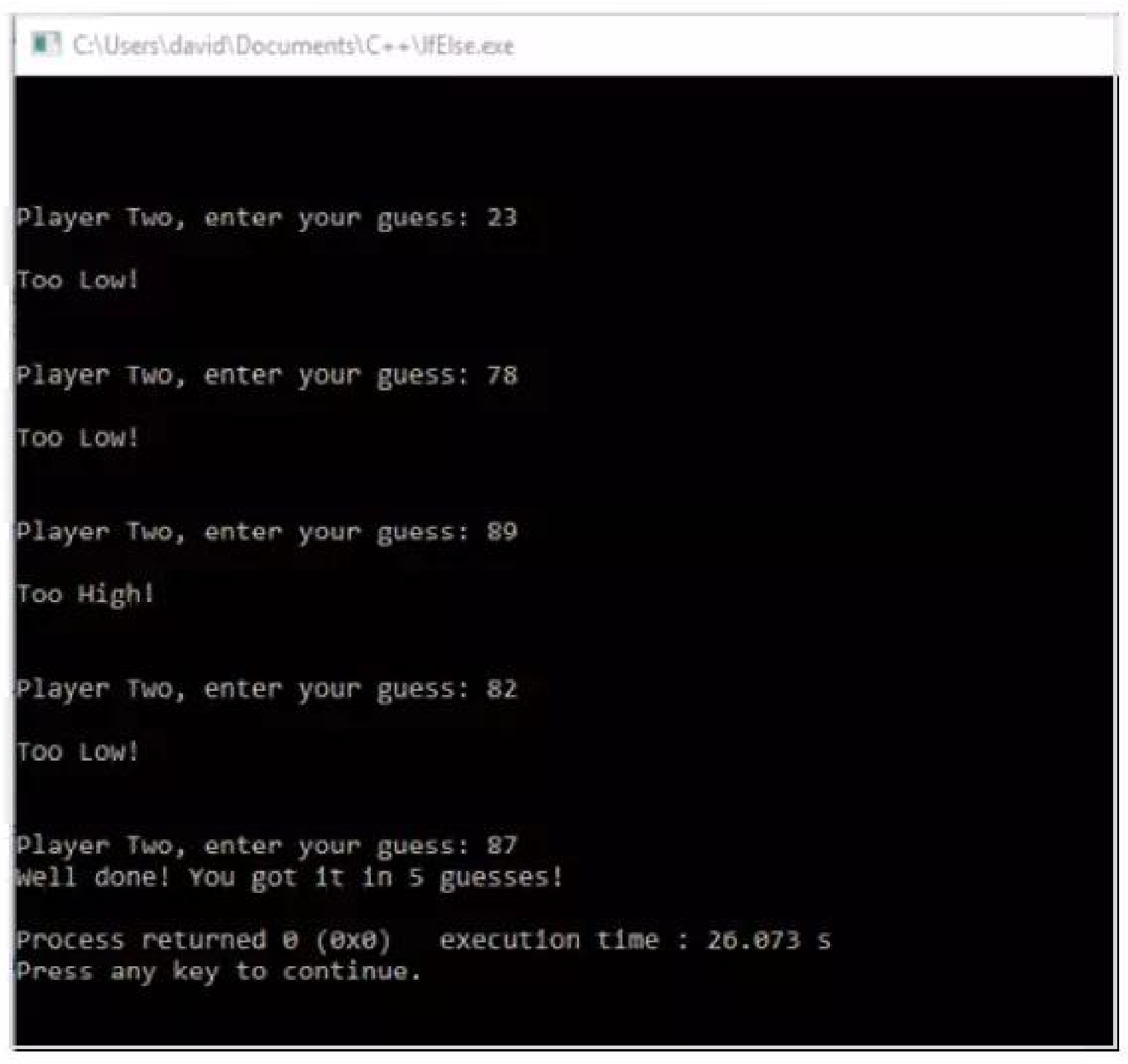
```
  do
    {
        cout << “\nPlayer Two, enter your guess: “;
        cin >> guess;
        tries++;
        if (guess > num)
        {
            cout << “\nToo High!\n” << endl;
        }
        else if (guess < num)
        {
            cout << “\nToo Low!\n” << endl;
        }
        else if (guess == num)
        {
            cout << “Well done! You got it in “ <<
tries << “ guesses!” << endl;
        }
    } while (guess != num);

    return 0;
```



```
using namespace std;

int main ()

{
    int num, guess, tries = 0;

    cout << "***** Two-player number guessing game ****" << endl;
    cout << "\nPlayer One, enter a number for Player Two to guess: " << endl;
    cin >> num;
    cout << string(50, '\n');

    do
    {
        cout << "\nPlayer Two, enter your guess: ";
        cin >> guess;
        tries++;
        if (guess > num)
        {
            cout << "\nToo High!\n" << endl;
        }
        else if (guess < num)
        {
            cout << "\nToo Low!\n" << endl;
        }
        else if (guess == num)
        {
            cout << "Well done! You got it in " << tries << " guesses!" << endl;
        }
    } while (guess != num);

    return 0;
```

STEP 10

Grab a second player, then build and run the code. Player One enters the number to be guessed, then Player Two can take as many guesses as they need to get the right number. Want to make it harder? Maybe use decimal numbers.



```
Player Two, enter your guess: 23

Too Low!


Player Two, enter your guess: 78

Too Low!


Player Two, enter your guess: 89

Too High!


Player Two, enter your guess: 82

Too Low!


Player Two, enter your guess: 87
Well done! You got it in 5 guesses!

Process returned 0 (0x0)   execution time : 26.073 s
Press any key to continue.
```

Working with Code

By now you should have the essential building blocks of how to program in both Python and C++. You can create code, store data, ask the user questions, offer them a choice and repeat functions until they prove true and provide the correct output. What next then?

This final section looks at some of the more common coding mistakes that Python and C++ beginners make and where you can turn to next to continue your programming adventure.

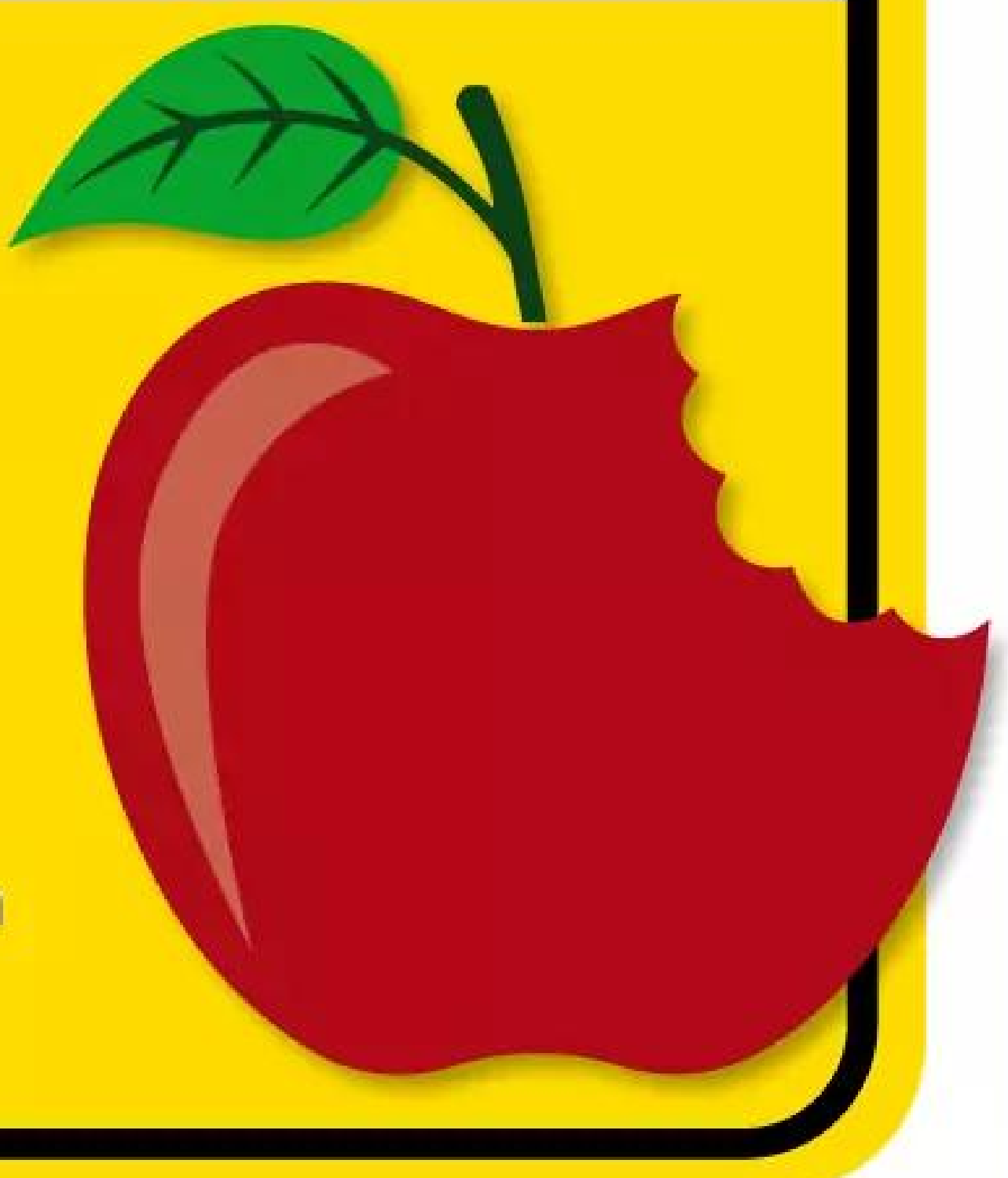
Common Coding Mistakes

When you start something new you're inevitably going to make mistakes, this is purely down to inexperience and those mistakes are great teachers in themselves. However, even experts make the occasional mishap. Thing is, to learn from them as best you can.

X=MISTAKE, PRINT Y

There are many pitfalls for the programmer to be aware of, far too many to be listed here. Being able to recognise a mistake and fix it is when you start to move into more advanced territory.

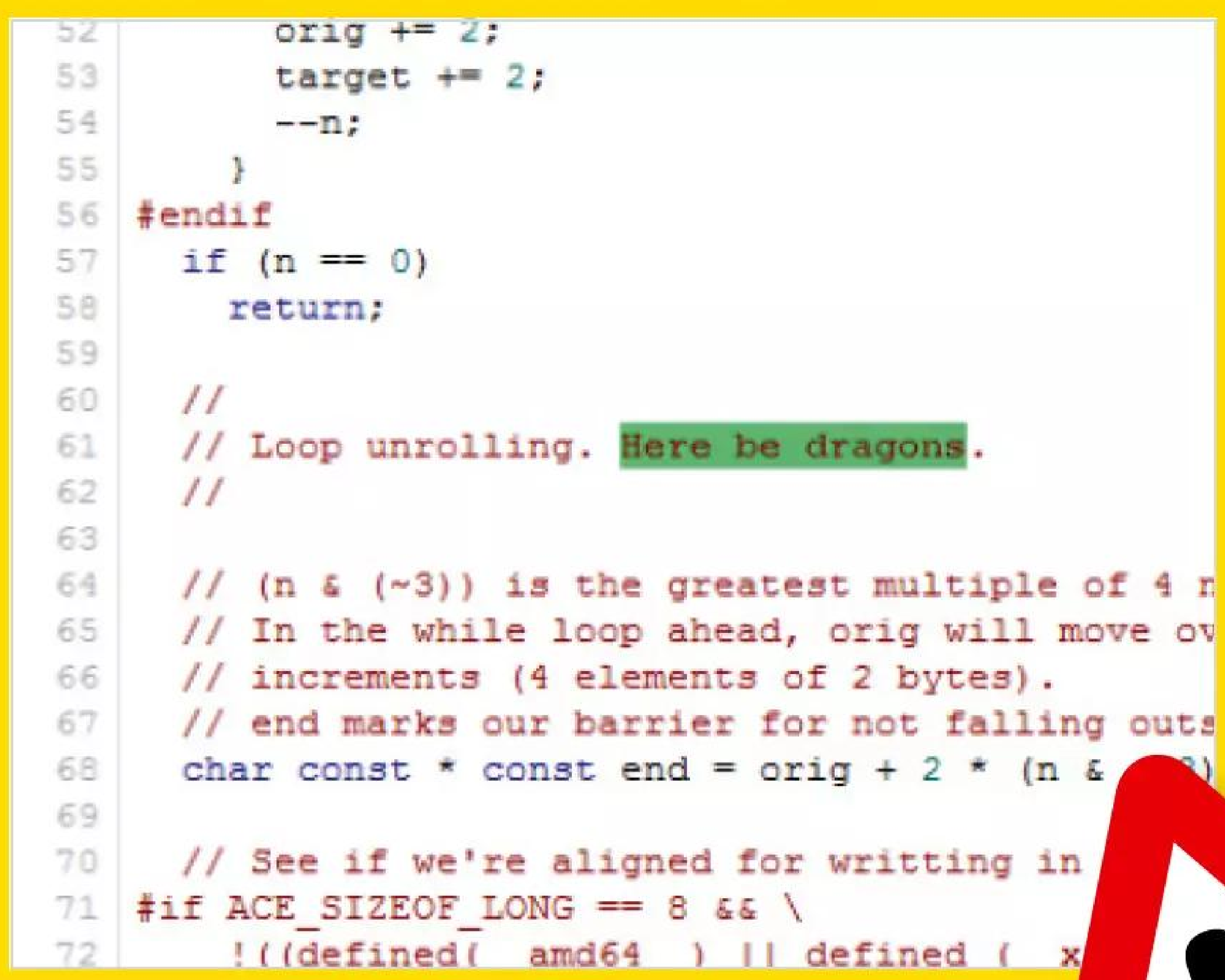
SMALL CHUNKS



It would be wonderful to be able to work like Neo from The Matrix movies. Simply ask, your operator loads it into your memory and you instantly know everything about the subject. Sadly though, we can't do that. The first major pitfall is someone trying to learn too much, too quickly. So take coding in small pieces and take your time.

//COMMENTS

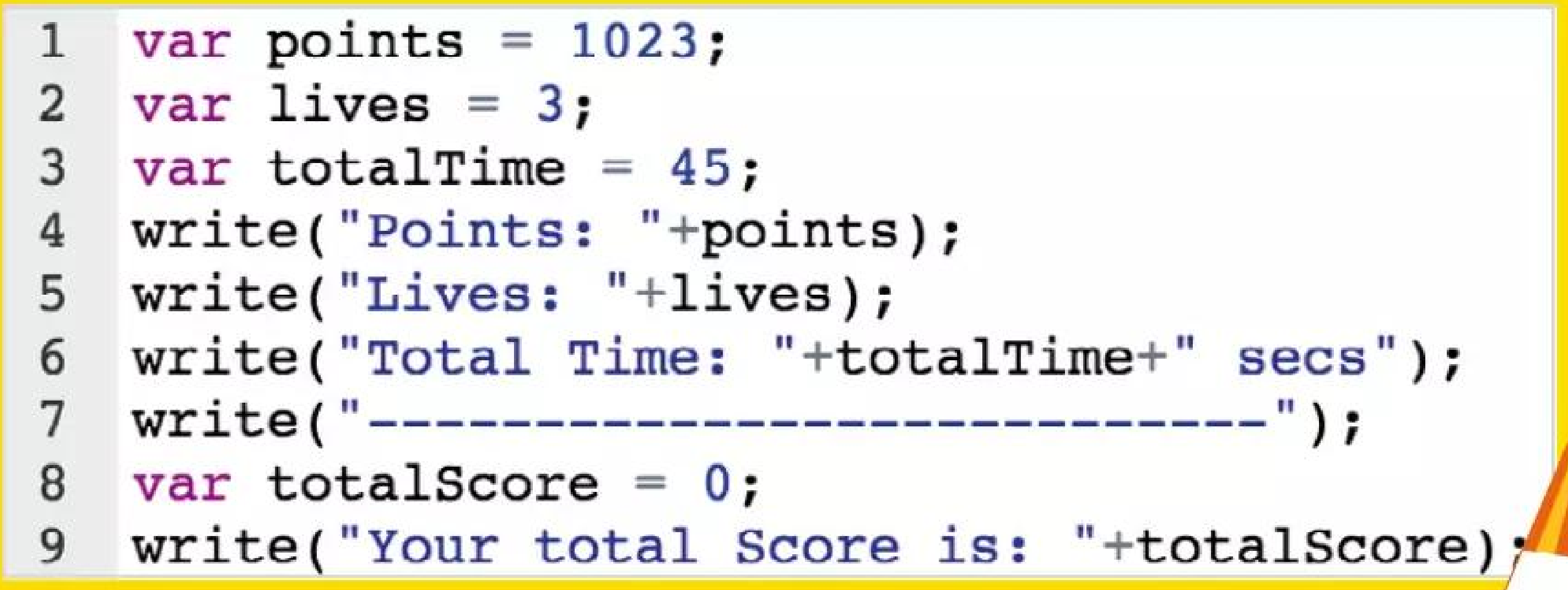
Use comments. It's a simple concept but commenting on your code saves so many problems when you next come to look over it. Inserting comment lines helps you quickly sift through the sections of code that are causing problems; also useful if you need to review an older piece of code.



```
        orig += 2;
        target += 2;
        --n;
      }
#endif
  if (n == 0)
    return;

  //
  // Loop unrolling. Here be dragons.
  //

  // (n & (~3)) is the greatest multiple of 4 n
  // In the while loop ahead, orig will move ov
  // increments (4 elements of 2 bytes).
  // end marks our barrier for not falling outs
  char const * const end = orig + 2 * (n &

  // See if we're aligned for writting in
#if ACE_SIZEOF_LONG == 8 && \
    !((defined(  amd64  ) || defined (  x
```

EASY VARIABLES

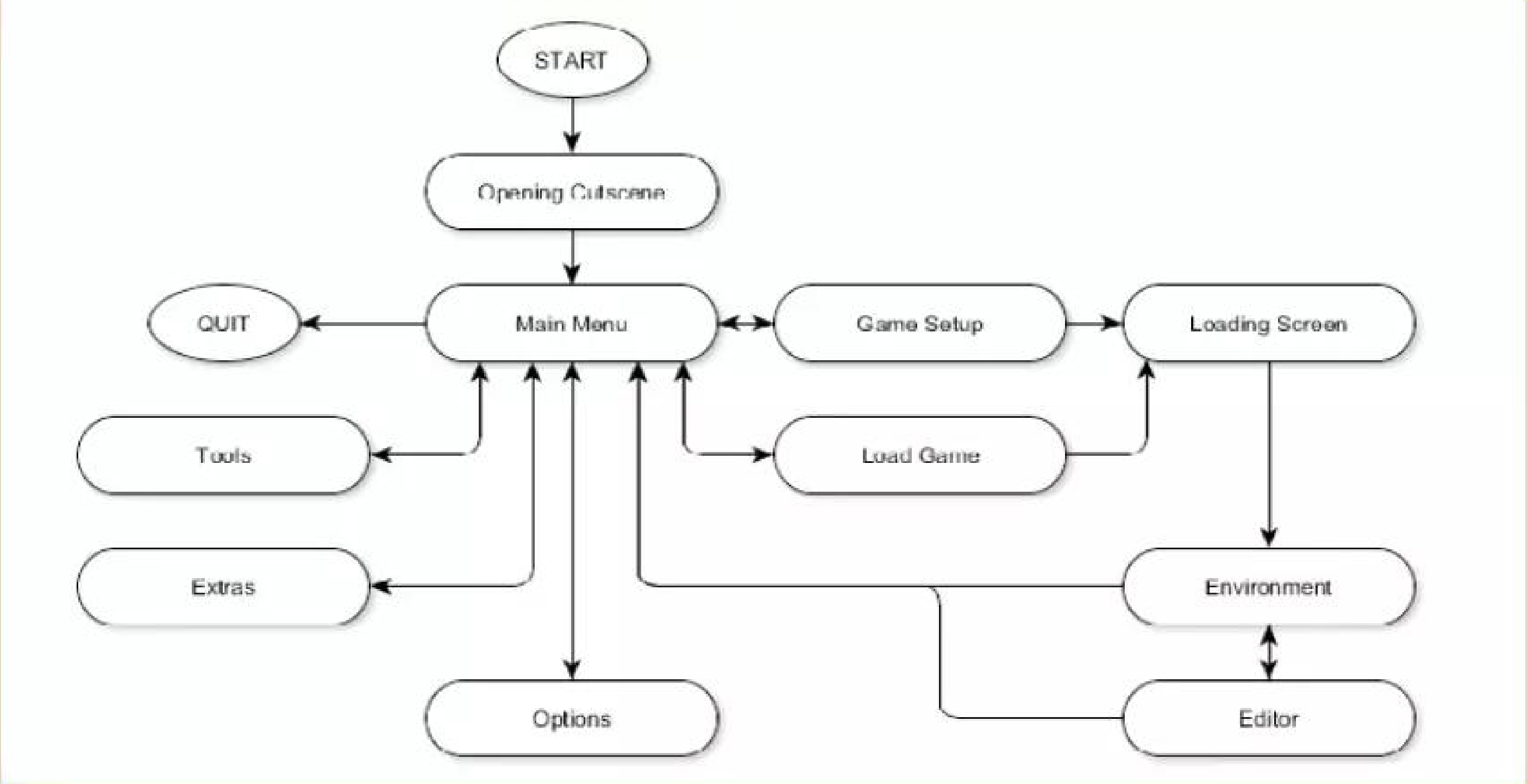
Meaningful naming for variables is a must to eliminate common coding mistakes. Having letters of the alphabet is fine but what happens when the code states there's a problem with x variable. It's not too difficult to name variables lives, money, player1 and so on.



```
var points = 1023;
var lives = 3;
var totalTime = 45;
write("Points: "+points);
write("Lives: "+lives);
write("Total Time: "+totalTime+" secs");
write("---------------------------");
var totalScore = 0;
write("Your total Score is: "+totalScore);
```

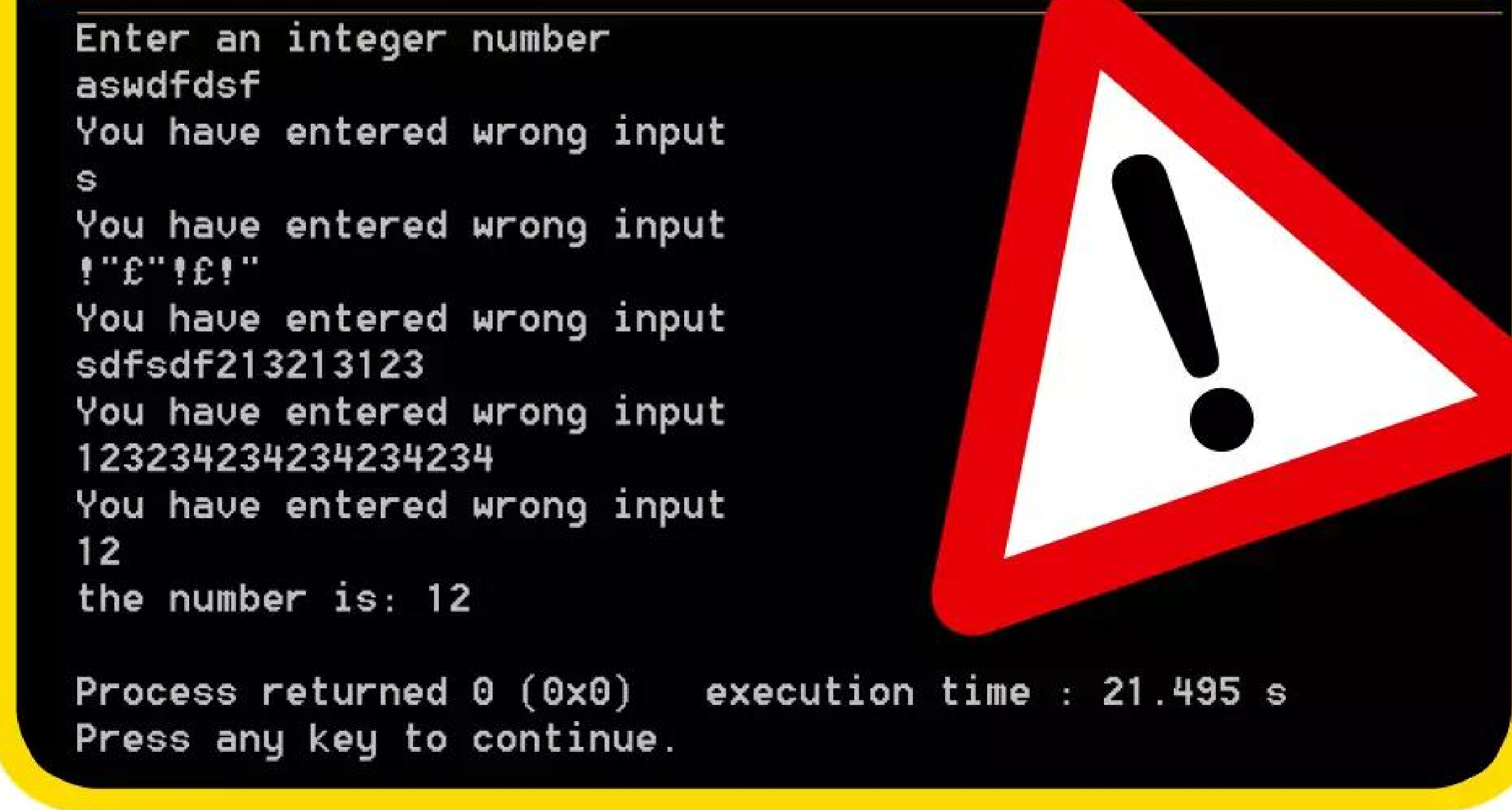
PLAN AHEAD

While it's great to wake up one morning and decide to code a classic text adventure, it's not always practical without a good plan. Small snippets of code can be written without too much thought and planning but longer and more in-depth code requires a good working plan to stick to and help iron out the bugs.



USER ERROR

User input is often a paralysing mistake in code. For example, when the user is supposed to enter a number for their age and instead they enter it in letters. Often a user can enter so much into an input that it overflows some internal buffer, thus sending the code crashing. Watch those user inputs and clearly state what's needed from them.



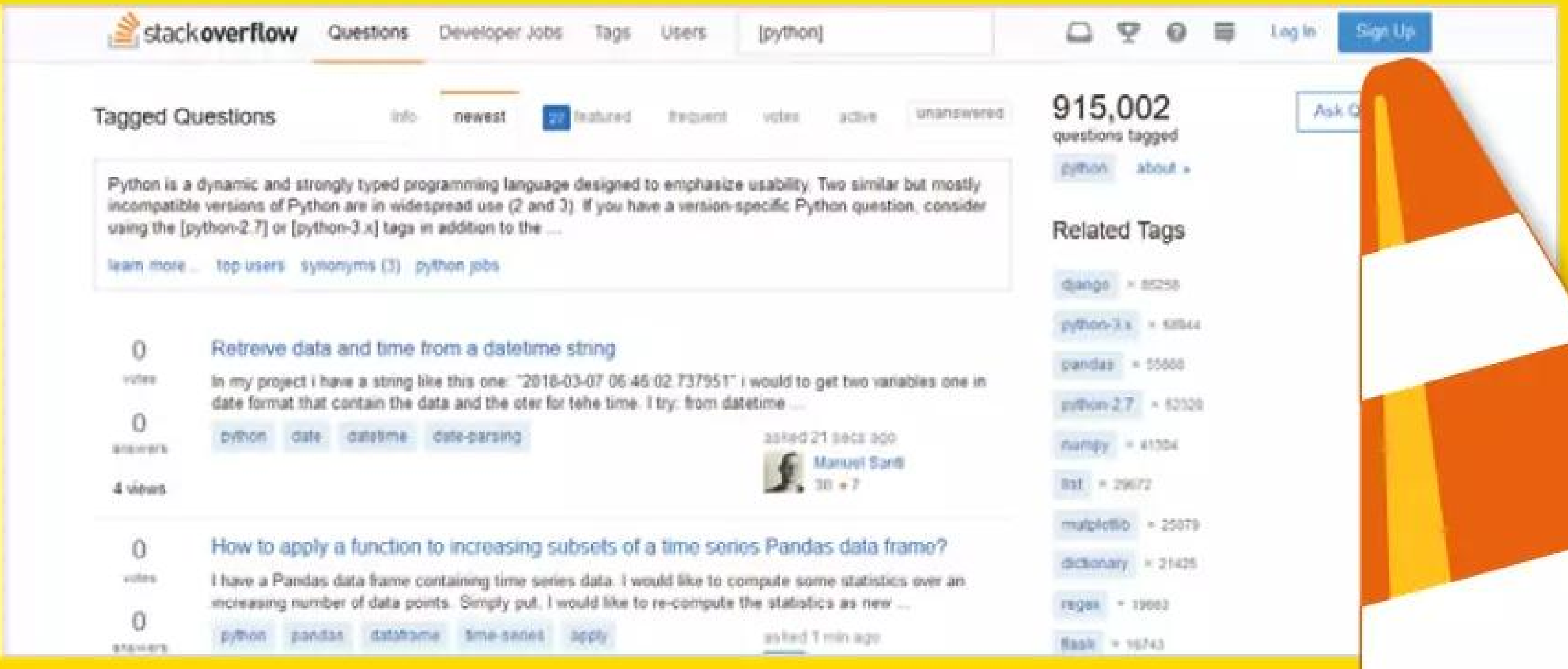
```
Enter an integer number
aswdfdsf
You have entered wrong input
s
You have entered wrong input
!"£"!£!"
You have entered wrong input
sdfsdf213213123
You have entered wrong input
12323423423423423
You have entered wrong input
12
the number is: 12

Process returned 0 (0x0)   execution time : 21.495 s
Press any key to continue.
```

RE-INVENTING WHEELS

You can easily spend days trying to fathom out a section of code to achieve a given result and it's frustrating and often time-wasting. While it's equally rewarding to solve the problem yourself, often the same code is out there on the Internet somewhere. Don't try and re-invent the wheel, look to see if some else has done it first.

HELP!

Asking for help is something most of us has struggled with in the past. Will the people we're asking laugh at us? Am I wasting everyone's time? It's a common mistake for someone to suffer in silence. However, as long as you ask the in the correct manner, obey any forum rules and be polite, then your question isn't silly.



Tagged Questions

Python is a dynamic and strongly typed programming language designed to emphasize usability. Two similar but mostly incompatible versions of Python are in widespread use (2 and 3). If you have a version-specific Python question, consider using the [python-2.7] or [python-3.x] tags in addition to the ...

learn more... top users synonyms (3) python jobs

0 votes 0 answers 4 views

Retreive data and time from a datetime string

In my project i have a string like this one: "2018-03-07 06:46:02.737951" i would to get two variables one in date format that contain the data and the oter for tehe time. I try: from datetime ...

python date datetime date-parsing

asked 21 secs ago Manuel Santi

0 votes 0 answers

How to apply a function to increasing subsets of a time series Pandas data frame?

I have a Pandas data frame containing time series data. I would like to compute some statistics over an increasing number of data points. Simply put, I would like to re-compute the statistics as new ...

python pandas dataframe time-series apply

asked 1 min ago

915,002 questions tagged

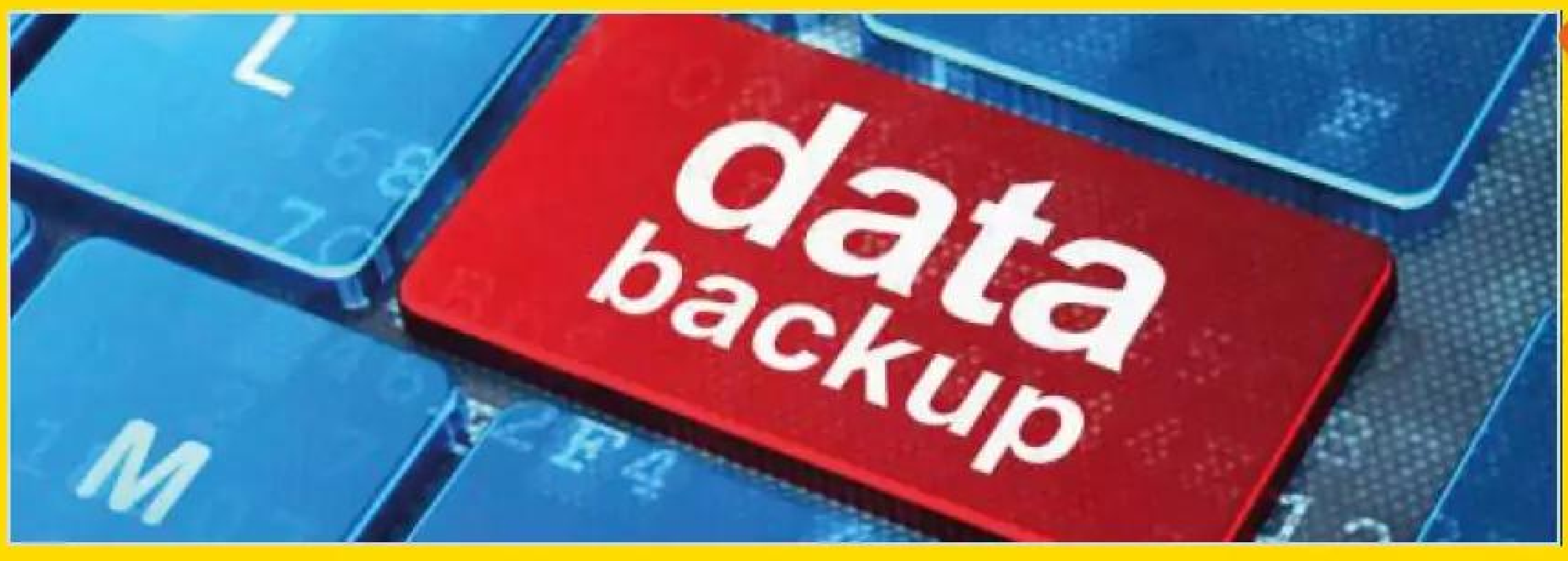
python about »

Related Tags

django × 85258
python-3.x × 68844
pandas × 55666
python-2.7 × 52320
numpy × 41304
list × 29672
matplotlib × 25079
dictionary × 21425
regex × 19662
flask × 16743

BACKUPS

Always make a backup of your work, with a secondary backup for any changes you've made. Mistakes can be rectified if there's a good backup in place to revert to for those times when something goes wrong. It's much easier to start where you left off, rather than starting from the beginning again.



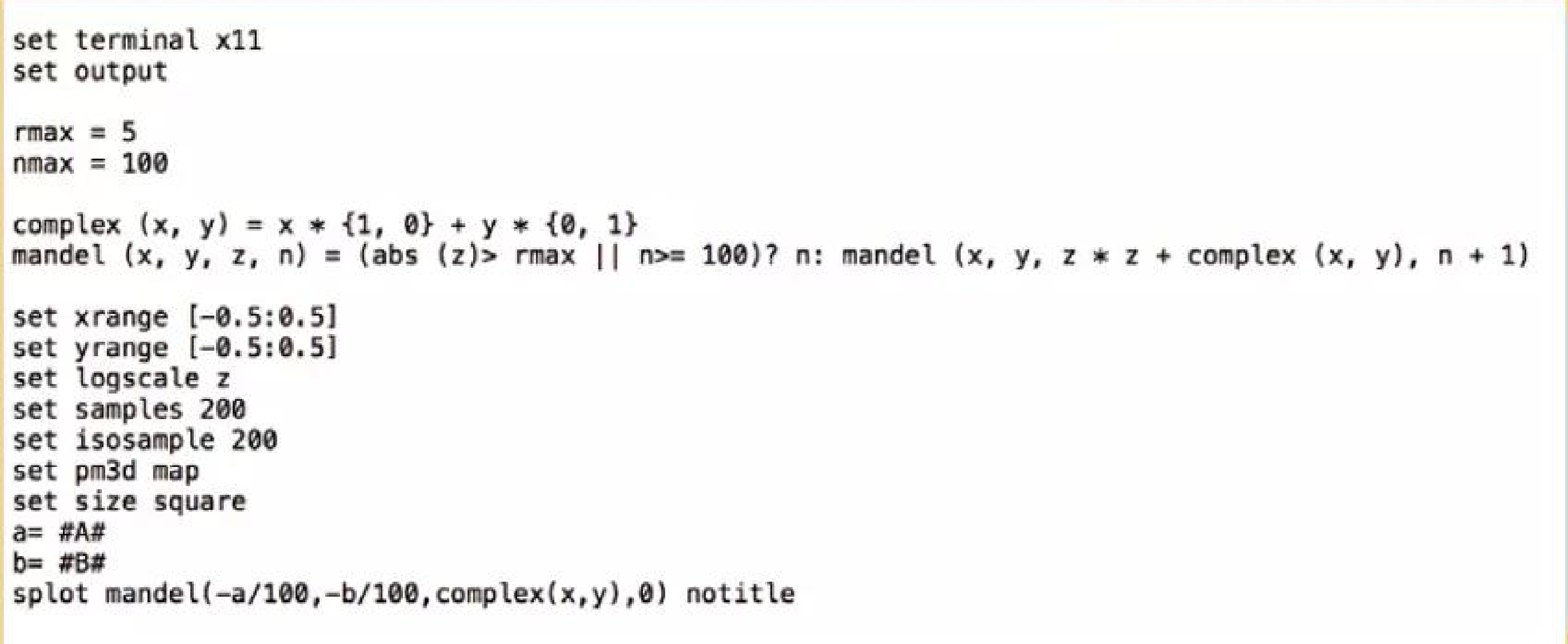
SECURE DATA

If you're writing code to deal with usernames and passwords, or other such sensitive data, then ensure that the data isn't in cleartext. Learn how to create a function to encrypt sensitive data, prior to feeding into a routine that can transmit or store it where someone may be able to get to view it.

ENCRYPT
data
security
secure
protection
privilege
unauthorized
hacker
device
computer

MATHS

If your code makes multiple calculations then you need to ensure that the maths behind it is sound. There are thousands of instances where programs have offered incorrect data based on poor mathematical coding, which can have disastrous effects depending on what the code is set to do. In short, double check your code equations.



```
set terminal x11
set output

rmax = 5
nmax = 100

complex (x, y) = x * {1, 0} + y * {0, 1}
mandel (x, y, z, n) = (abs (z)> rmax || n>= 100)? n: mandel (x, y, z * z + complex (x, y), n + 1)

set xrange [-0.5:0.5]
set yrange [-0.5:0.5]
set logscale z
set samples 200
set isosample 200
set pm3d map
set size square
a= #A#
b= #B#
splot mandel(-a/100,-b/100,complex(x,y),0) notitle
```

Beginner Python Mistakes

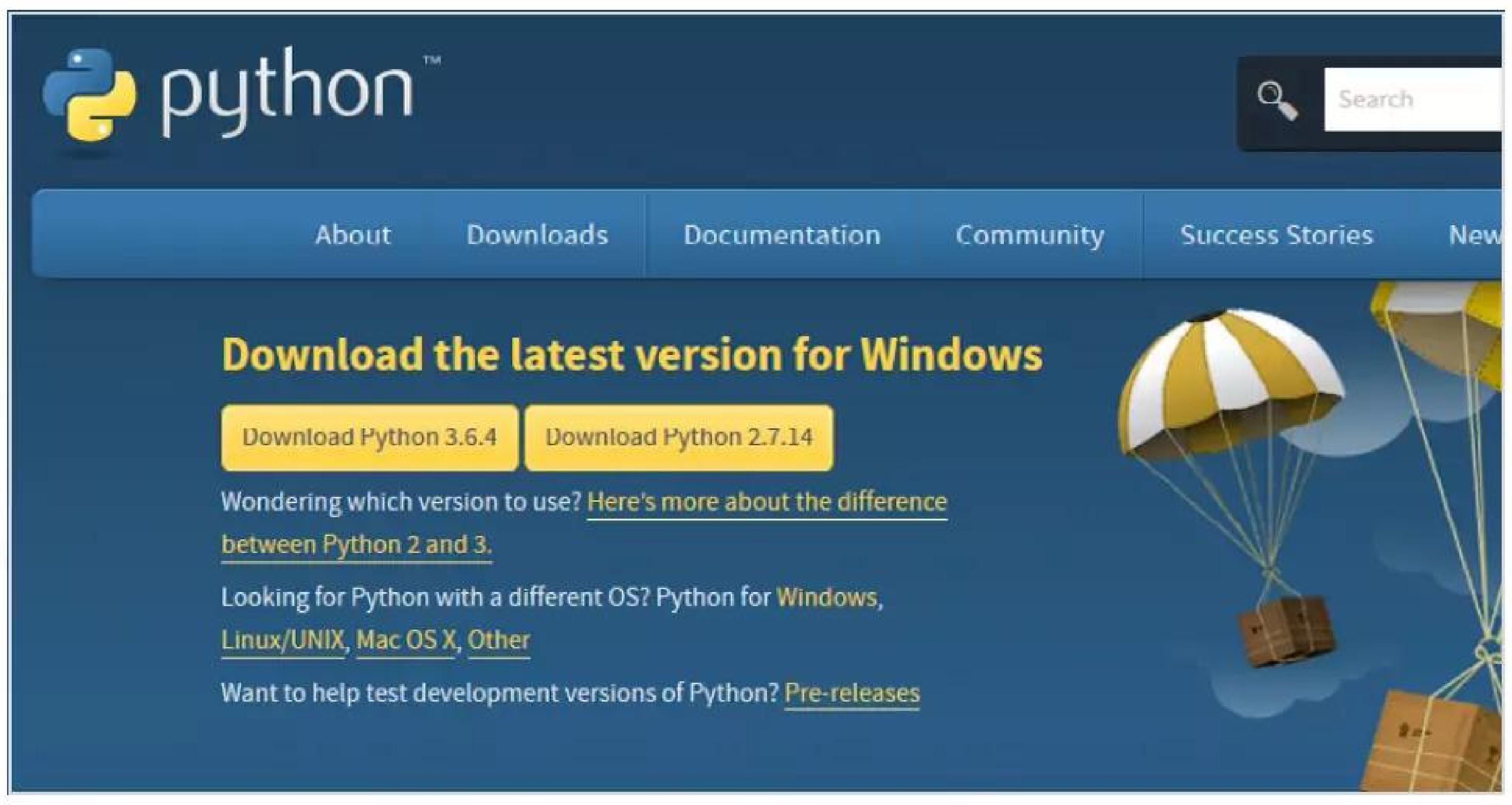
Python is a relatively easy language to get started in where there's plenty of room for the beginner to find their programming feet. However, as with any other programming language, it can be easy to make common mistakes that'll stop your code from running.

DEF BEGINNER(MISTAKES=10)

Here are ten common Python programming mistakes most beginners find themselves making. Being able to identify these mistakes will save you headaches in the future.

VERSIONS

To add to the confusion that most beginners already face when coming into programming, Python has two live versions of its language available to download and use. There is Python version 2.7.x and Python 3.6.x. The 3.6.x version is the most recent, and the one we'd recommend starting. But, version 2.7.x code doesn't always work with 3.6.x code and vice versa.



Download the latest version for Windows

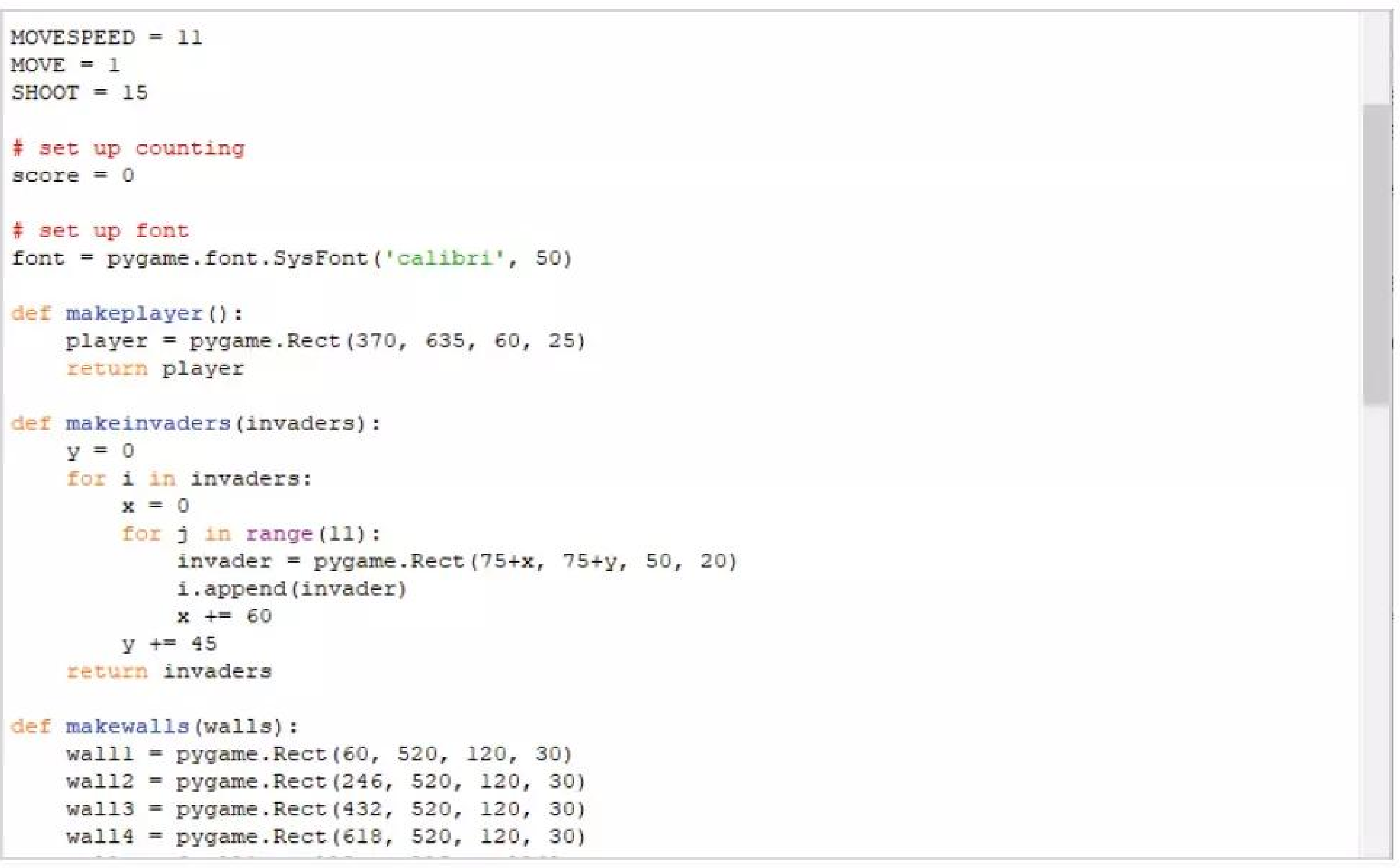
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

INDENTS, TABS AND SPACES

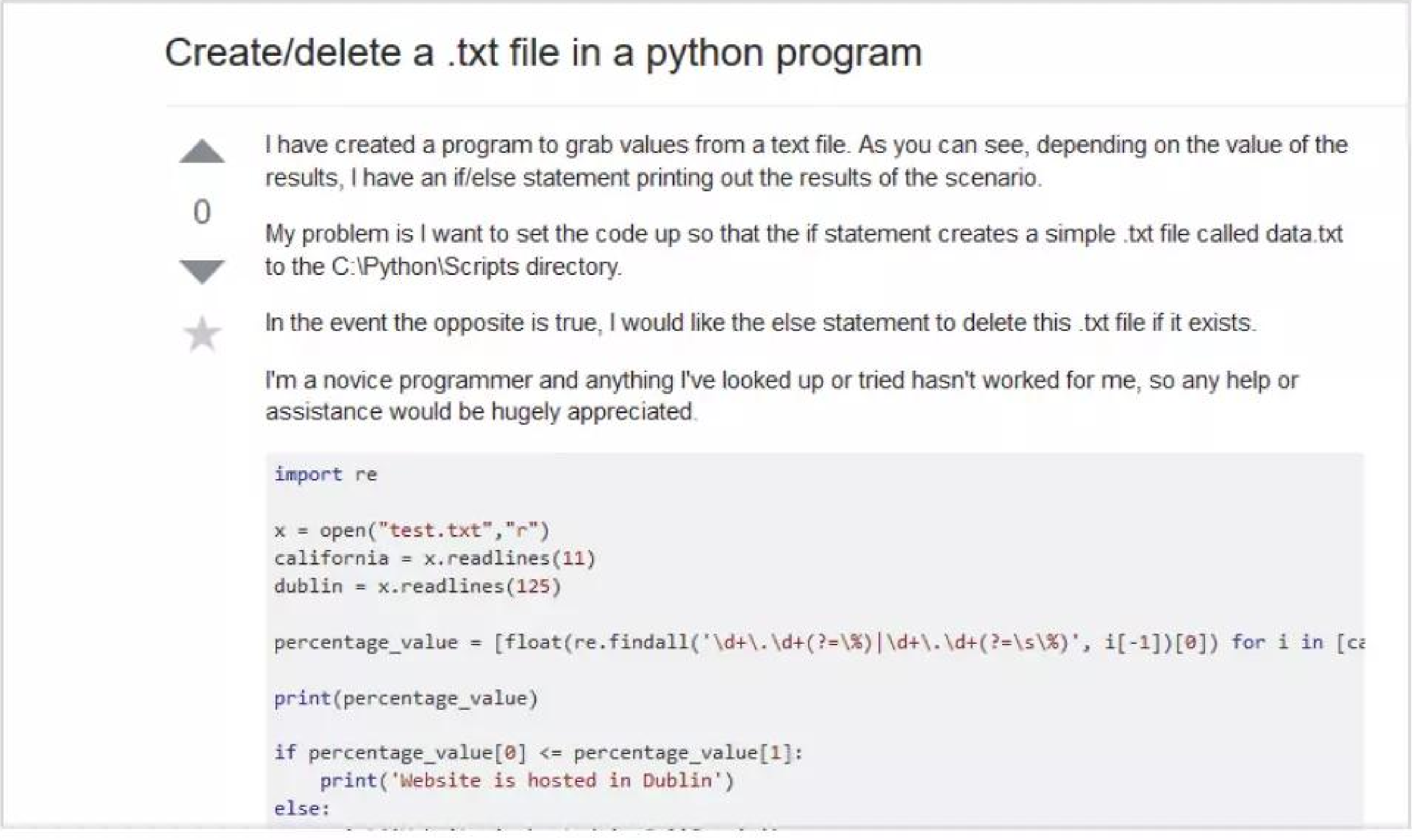
Python uses precise indentations when displaying its code. The indents mean that the code in that section is a part of the previous statement, and not something linked with another part of the code. Use four spaces to create an indent, not the Tab key.



```
MOVESPEED = 11
MOVE = 1
SHOOT = 15

# set up counting
score = 0

# set up font
font = pygame.font.SysFont('calibri', 50)

def makeplayer():
    player = pygame.Rect(370, 635, 60, 25)
    return player

def makeinvaders(invaders):
    y = 0
    for i in invaders:
        x = 0
        for j in range(11):
            invader = pygame.Rect(75+x, 75+y, 50, 20)
            i.append(invader)
            x += 60
        y += 45
    return invaders

def makewalls(walls):
    wall1 = pygame.Rect(60, 520, 120, 30)
    wall2 = pygame.Rect(246, 520, 120, 30)
    wall3 = pygame.Rect(432, 520, 120, 30)
    wall4 = pygame.Rect(618, 520, 120, 30)
```

THE INTERNET

Every programmer has and does at some point go on the Internet and copy some code to insert into their own routines. There's nothing wrong with using others' code, but you need to know how the code works and what it does before you go blindly running it on your own computer.



Create/delete a .txt file in a python program

I have created a program to grab values from a text file. As you can see, depending on the value of the results, I have an if/else statement printing out the results of the scenario.

My problem is I want to set the code up so that the if statement creates a simple .txt file called data.txt to the C:\Python\Scripts directory.

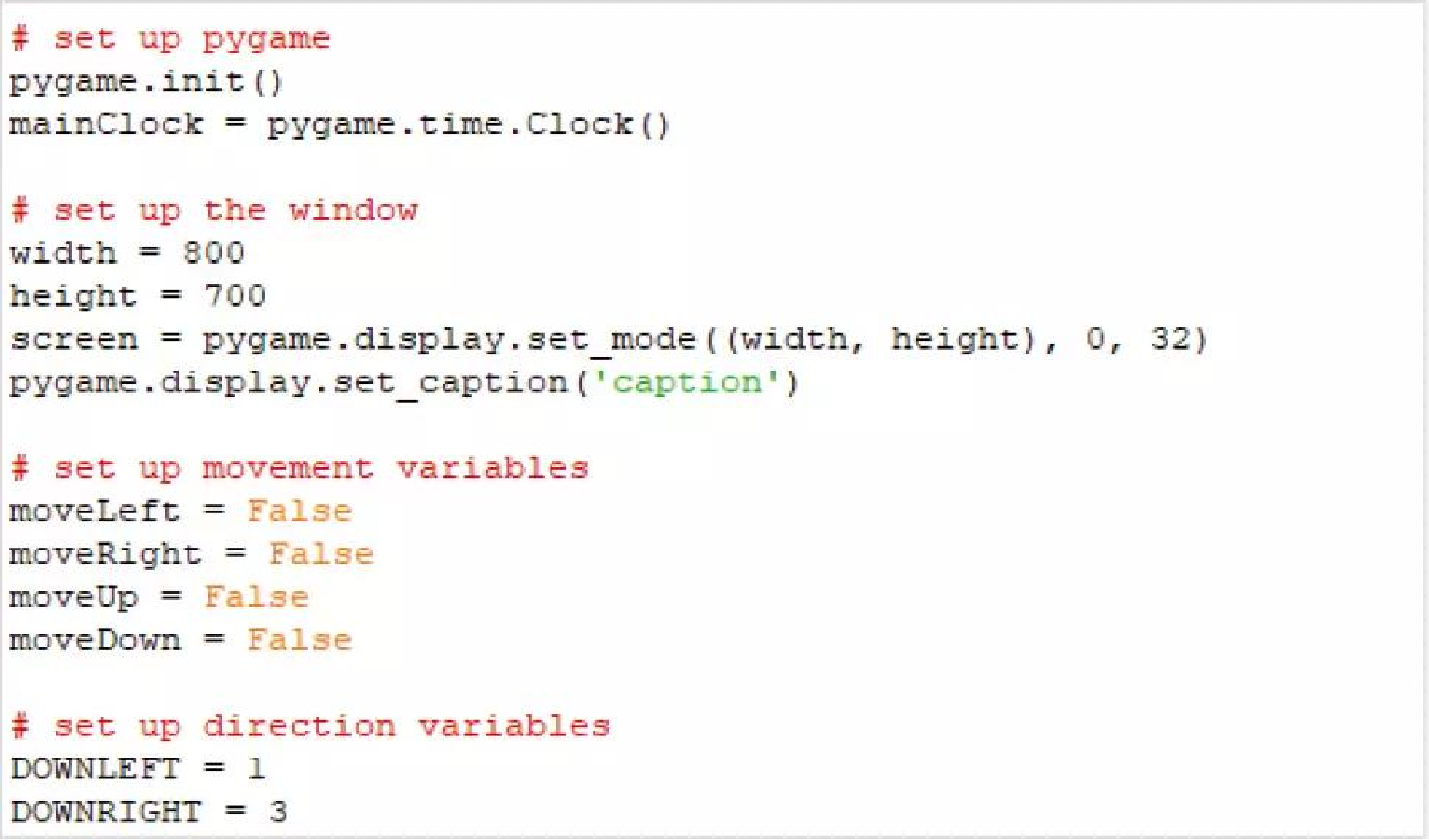
In the event the opposite is true, I would like the else statement to delete this .txt file if it exists.

I'm a novice programmer and anything I've looked up or tried hasn't worked for me, so any help or assistance would be hugely appreciated.

```
import re

x = open("test.txt","r")
california = x.readlines(11)
dublin = x.readlines(125)

percentage_value = [float(re.findall('\d+\.\d+(?=\%)|\d+\.\d+(?=\s\%)', i[-1])[0]) for i in [ca

print(percentage_value)

if percentage_value[0] <= percentage_value[1]:
    print('Website is hosted in Dublin')
else:
```

COMMENTING

Again we mention commenting. It's a hugely important factor in programming, even if you're the only one who is ever going to view the code, you need to add comments as to what's going on. Is this function where you lose a life? Write a comment and help you, or anyone else, see what's going on.



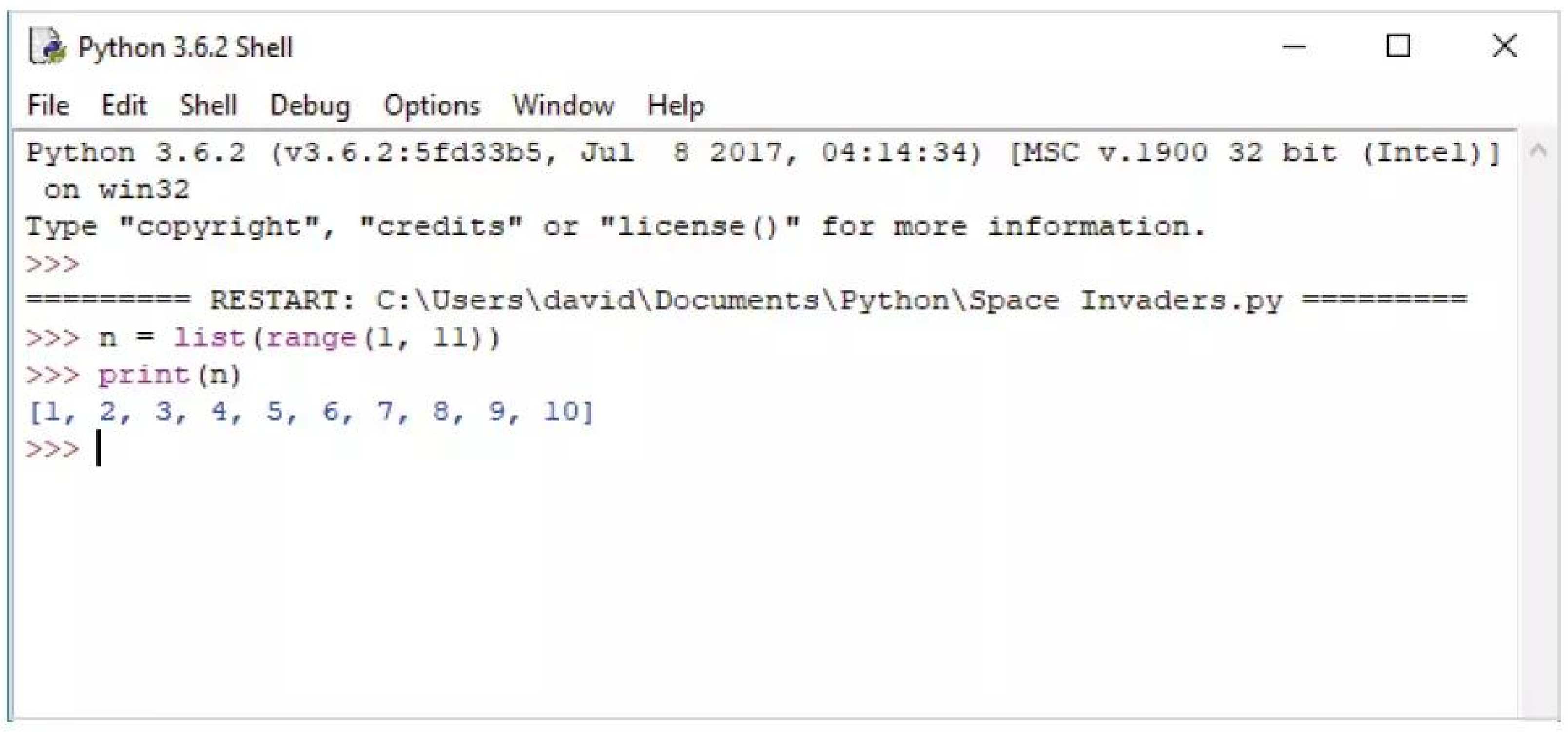
```
# set up pygame
pygame.init()
mainClock = pygame.time.Clock()

# set up the window
width = 800
height = 700
screen = pygame.display.set_mode((width, height), 0, 32)
pygame.display.set_caption('caption')

# set up movement variables
moveLeft = False
moveRight = False
moveUp = False
moveDown = False

# set up direction variables
DOWNLEFT = 1
DOWNRIGHT = 3
```

COUNTING LOOPS

Remember that in Python a loop doesn't count the last number you specify in a range. So if you wanted the loop to count from 1 to 10, then you will need to use:

`n = list(range(1, 11))`

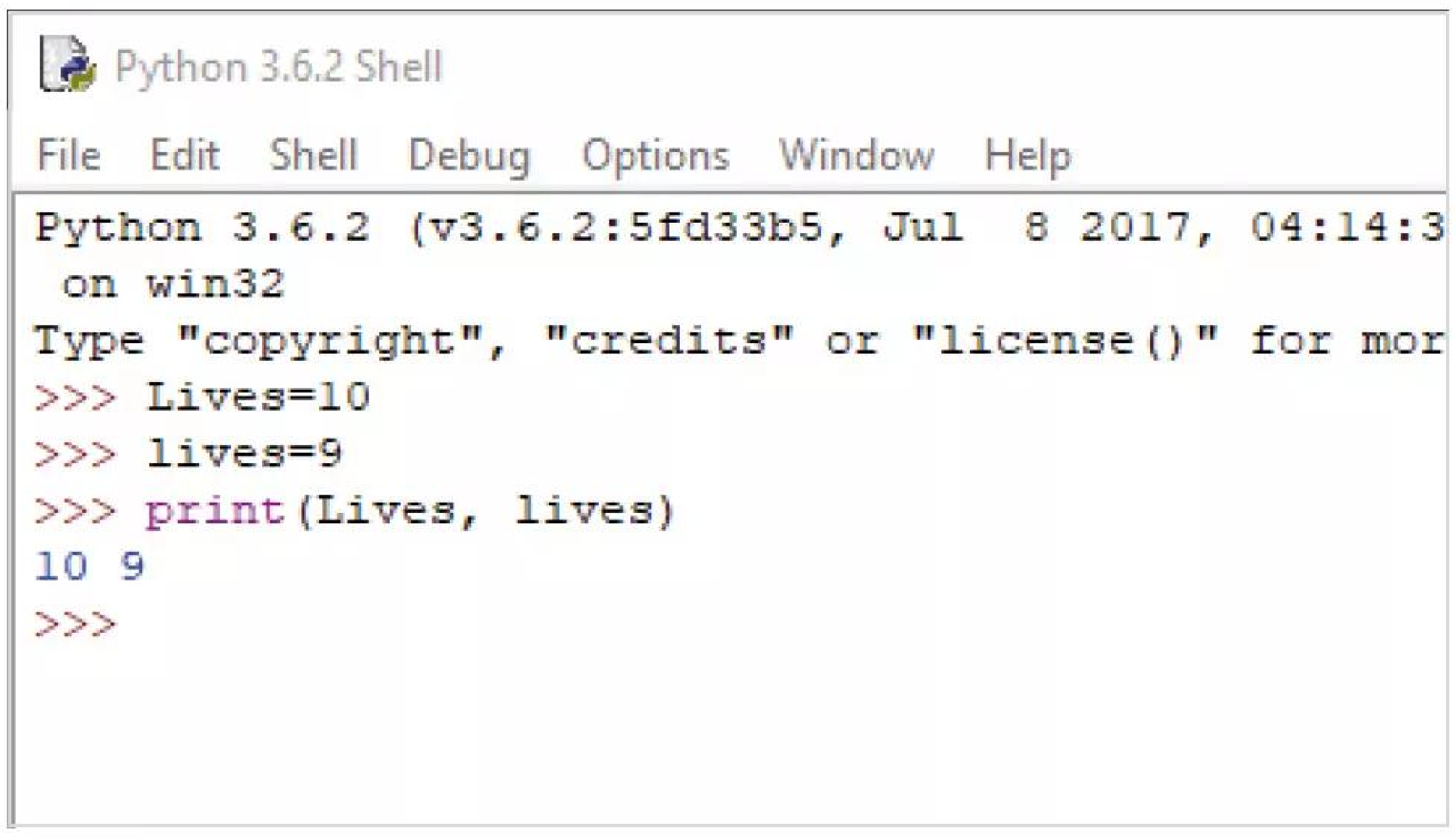
Which will return 1 to 10.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\david\Documents\Python\Space Invaders.py =========
>>> n = list(range(1, 11))
>>> print(n)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> 
```

CASE SENSITIVE

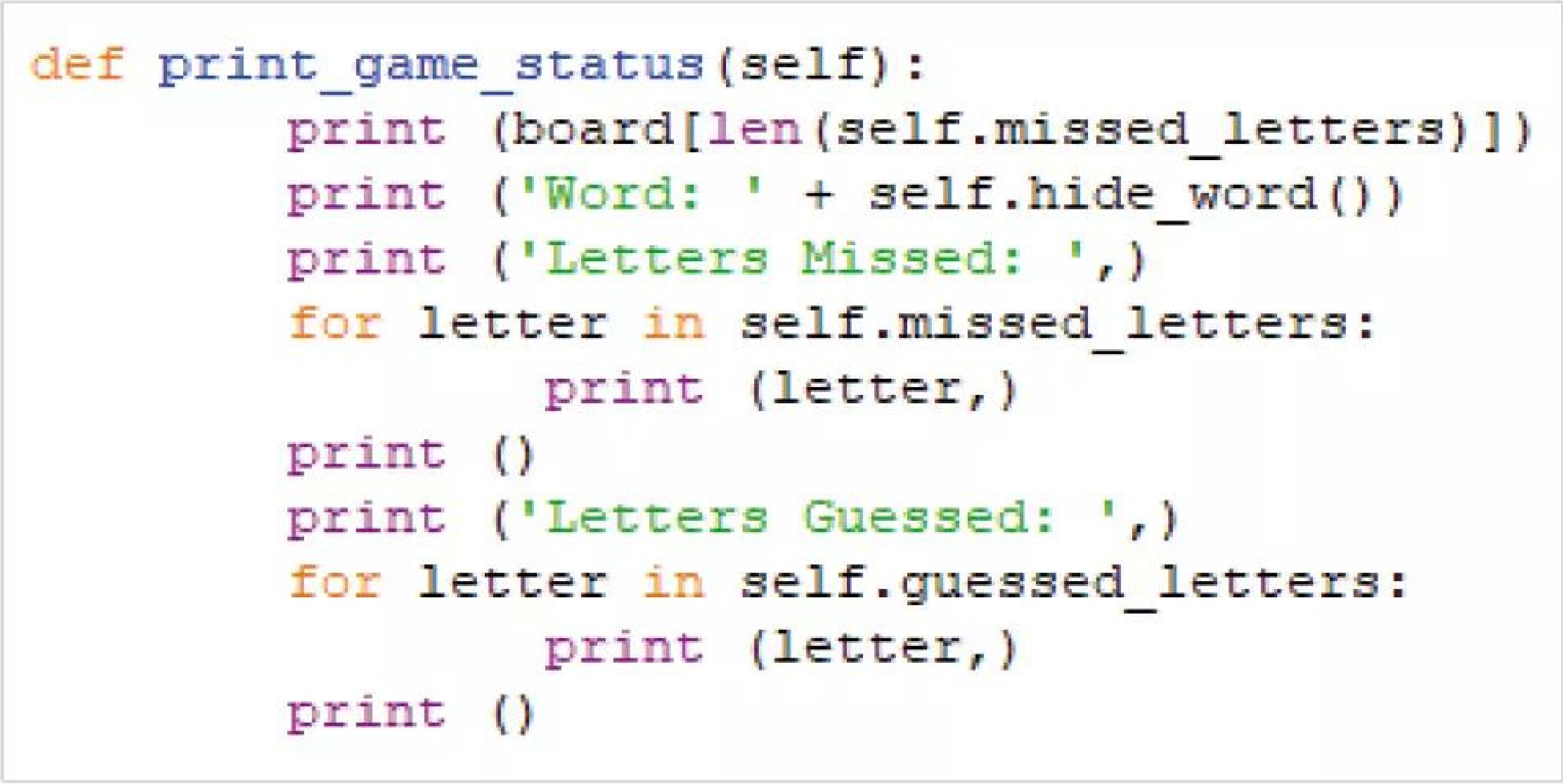
Python is a case sensitive programming language, so you will need to check any variables you assign. For example, `Lives=10` is a different variable to lives=10, calling the wrong variable in your code can have unexpected results.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:3
 on win32
Type "copyright", "credits" or "license()" for mor
>>> Lives=10
>>> lives=9
>>> print(Lives, lives)
10 9
>>>
```

BRACKETS

Everyone forgets to include that extra bracket they should have added to the end of the statement. Python relies on the routine having an equal amount of closed brackets to open brackets, so any errors in your code could be due to you forgetting to count your brackets; including square brackets.



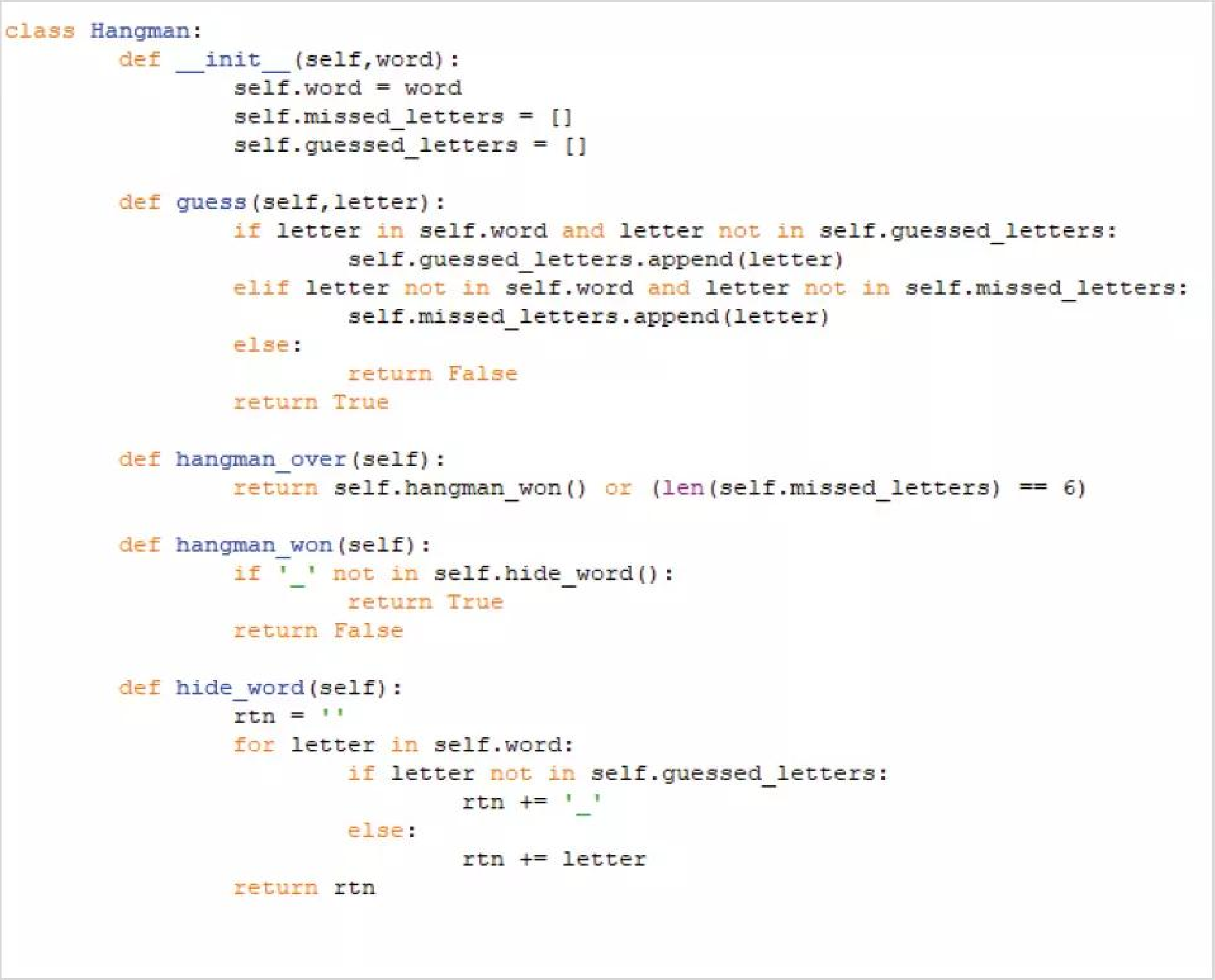
```
def print_game_status(self):
        print (board[len(self.missed_letters)])
        print ('Word: ' + self.hide_word())
        print ('Letters Missed: ',)
        for letter in self.missed_letters:
                print (letter,)
        print ()
        print ('Letters Guessed: ',)
        for letter in self.guessed_letters:
                print (letter,)
        print ()
```

COLONS

It's common for beginners to forget to add a colon to the end of a structural statement, such as:

```
class Hangman:
def guess(self, letter):
```

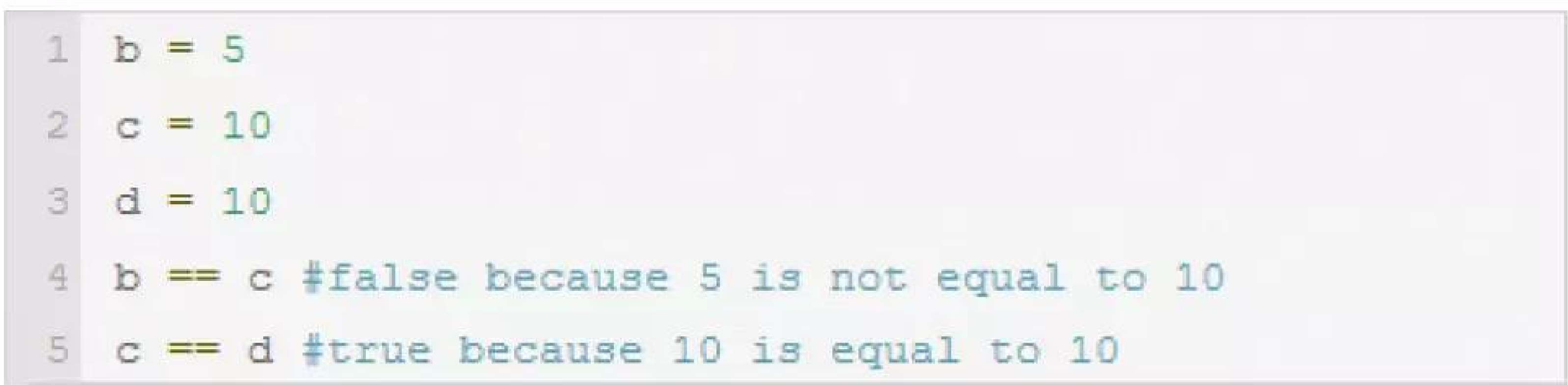
And so on. The colon is what separates the code, and creates the indents to which the following code belongs to.



```
class Hangman:
        def __init__(self,word):
                self.word = word
                self.missed_letters = []
                self.guessed_letters = []

        def guess(self,letter):
                if letter in self.word and letter not in self.guessed_letters:
                        self.guessed_letters.append(letter)
                elif letter not in self.word and letter not in self.missed_letters:
                        self.missed_letters.append(letter)
                else:
                        return False
                return True

        def hangman_over(self):
                return self.hangman_won() or (len(self.missed_letters) == 6)

        def hangman_won(self):
                if '_' not in self.hide_word():
                        return True
                return False

        def hide_word(self):
                rtn = ''
                for letter in self.word:
                        if letter not in self.guessed_letters:
                                rtn += '_'
                        else:
                                rtn += letter
                return rtn
```

OPERATORS

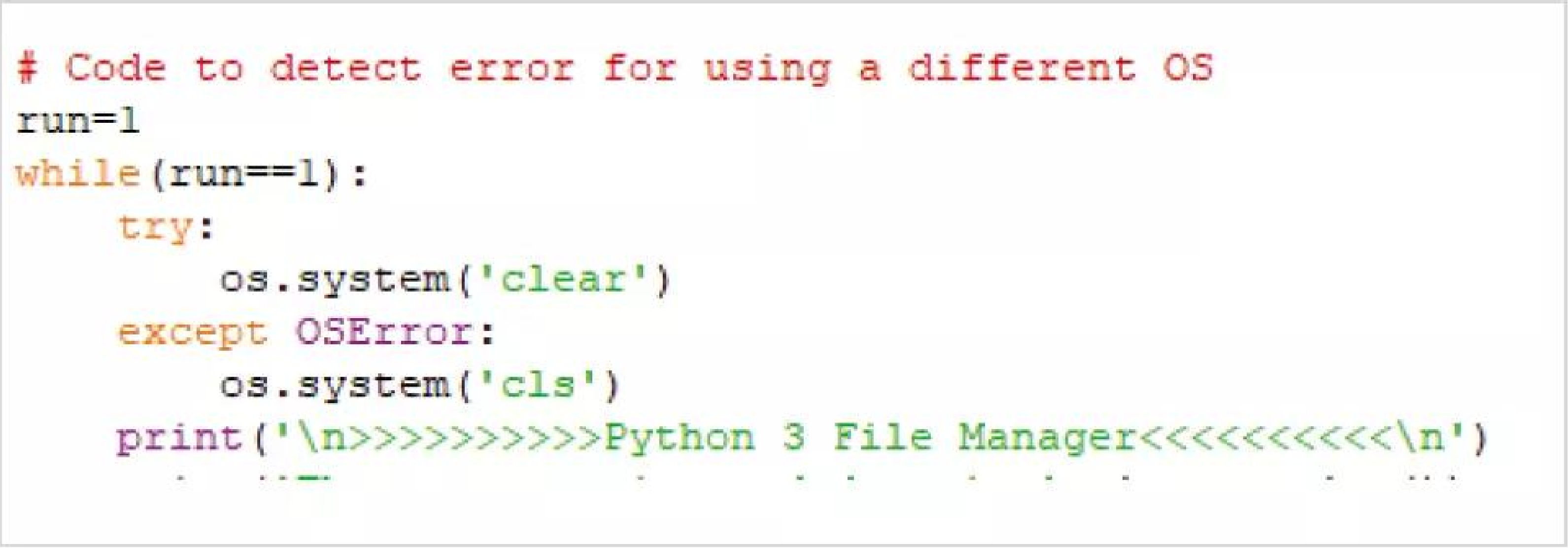
Using the wrong operator is also a common mistake to make. When you're performing a comparison between two values, for example, you need to use the equality operator (a double equals, ==). Using a single equal (=) is an assignment operator that places a value to a variable (such as, lives=10).



```
b = 5
c = 10
d = 10
b == c #false because 5 is not equal to 10
c == d #true because 10 is equal to 10
```

OPERATING SYSTEMS

Writing code for multiple platforms is difficult, especially when you start to utilise the external commands of the operating system. For example, if your code calls for the screen to be cleared, then for Windows you would use `cls`. Whereas, for Linux you need to use `clear`. You need to solve this by capturing the error and issuing it with an alternative command.



```
# Code to detect error for using a different OS
run=1
while(run==1):
    try:
        os.system('clear')
    except OSError:
        os.system('cls')
    print('\n>>>>>>>>>>Python 3 File Manager<<<<<<<<<<\n')
```

Beginner C++ Mistakes

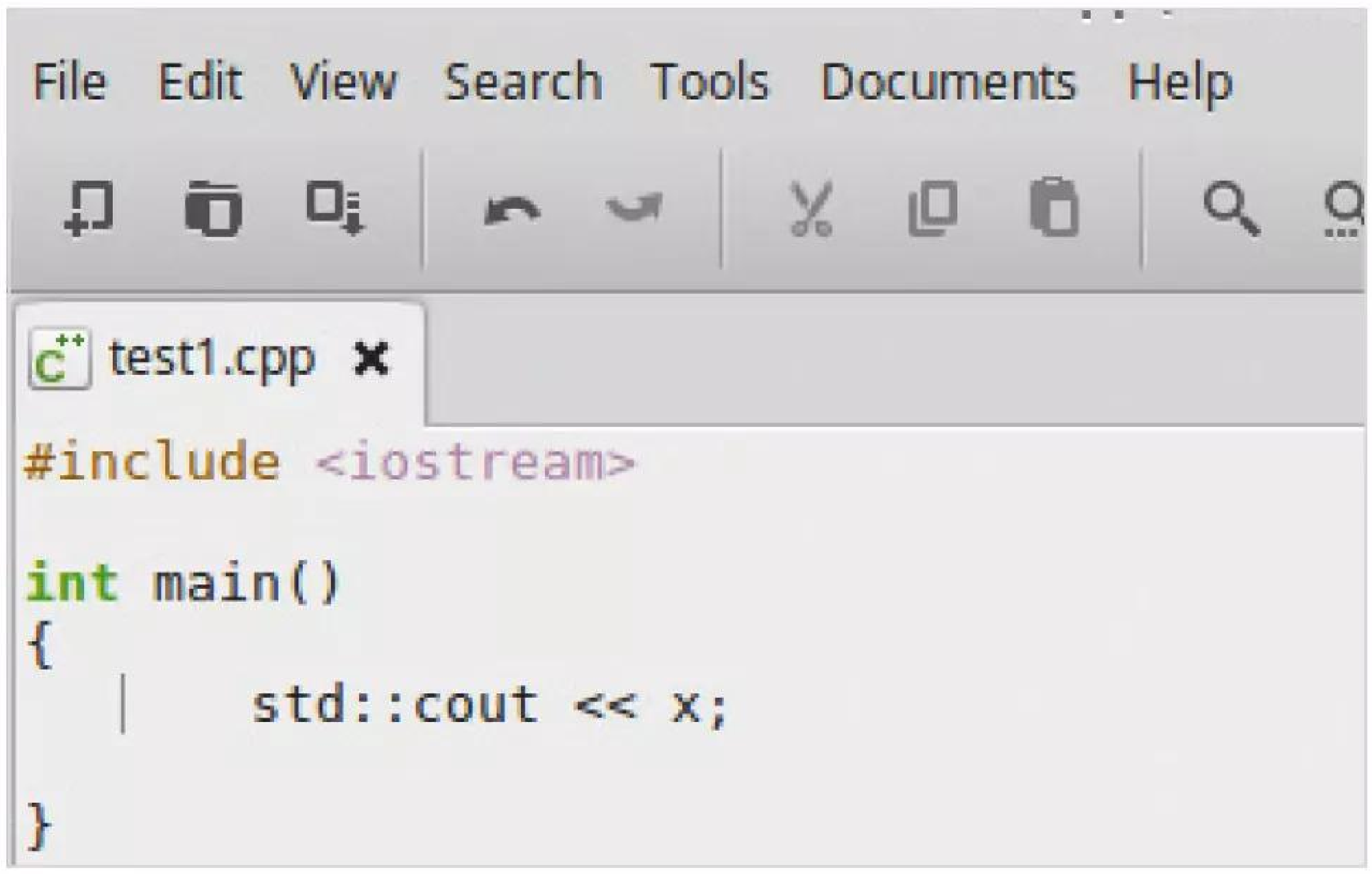
There are many pitfalls the C++ developer can encounter, especially as this is a more complex and often unforgiving language to master. Beginners need to take C++ a step at a time and digest what they've learned before moving on.

VOID(C++, MISTAKES)

Admittedly it's not just C++ beginners that make the kinds of errors we outline on these pages, even hardened coders are prone to the odd mishap here and there. Here are some common issues to try and avoid.

UNDECLARED IDENTIFIERS

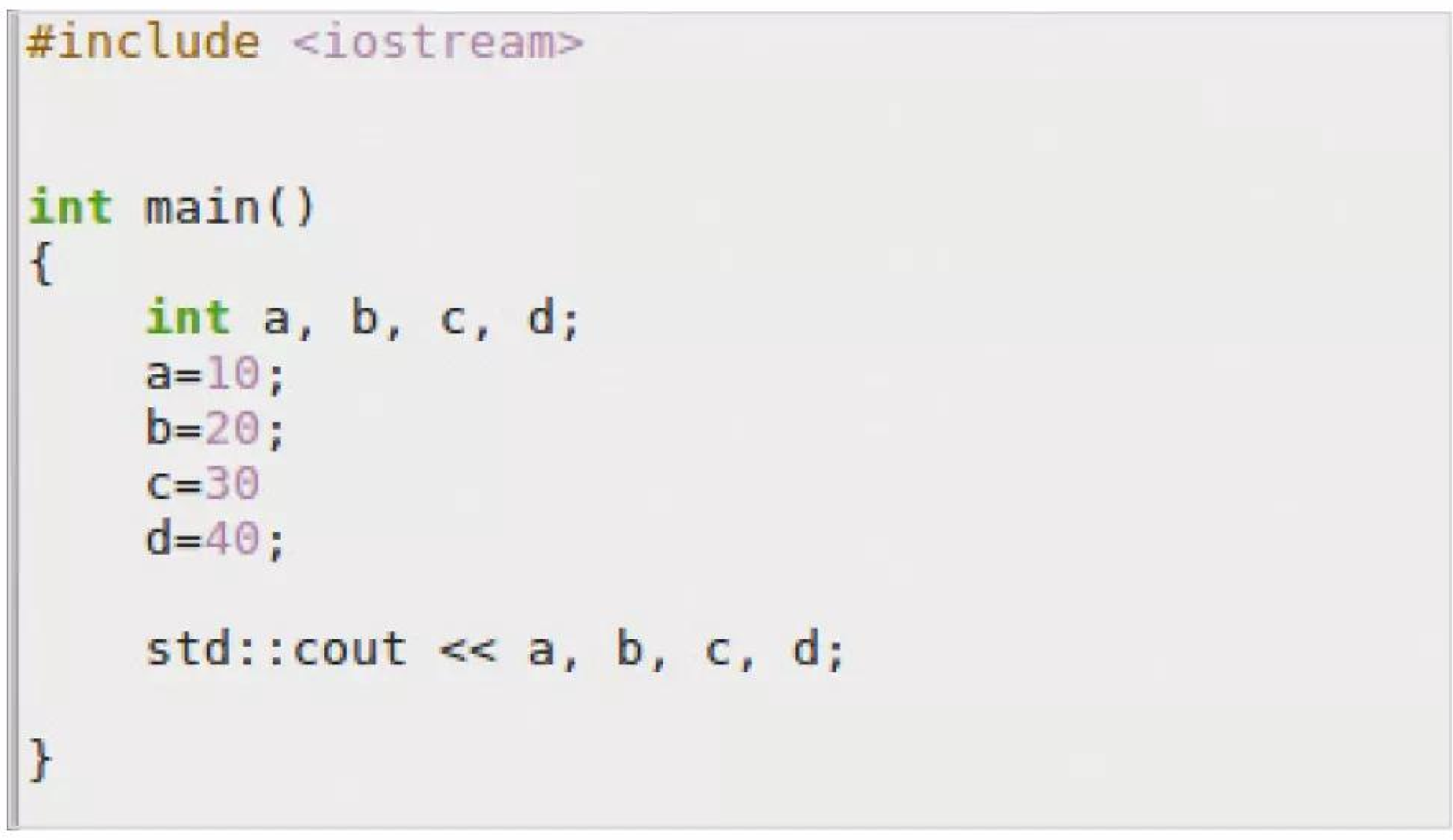
A common C++ mistake, and to be honest a common mistake with most programming languages, is when you try and output a variable that doesn't exist. Displaying the value of x on-screen is fine but not if you haven't told the compiler what the value of x is to begin with.



```
#include <iostream>

int main()
{
        std::cout << x;

}
```

SEMICOLONS

Remember that each line of a C++ program must end with a semicolon. If it doesn't then the compiler treats the line with the missing semicolon as the same line with the next semicolon on. This creates all manner of problems when trying to compile, so don't forget those semicolons.



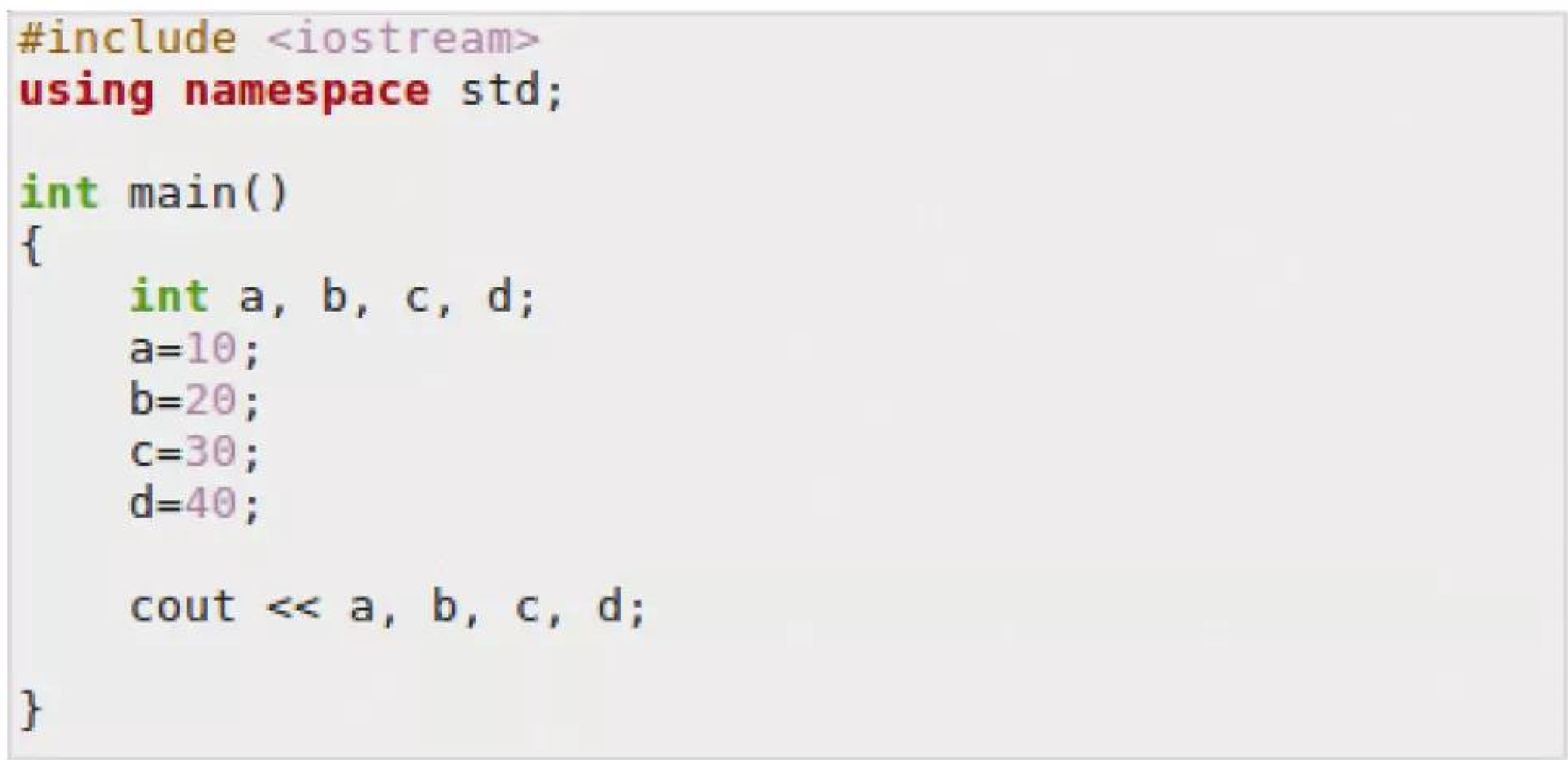
```
#include <iostream>


int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30
    d=40;

    std::cout << a, b, c, d;

}
```

STD NAMESPACE

Referencing the Standard Library is common for beginners throughout their code, but if you miss the std:: element of a statement, your code errors out when compiling. You can combat this by adding:

```
using namespace std;
```

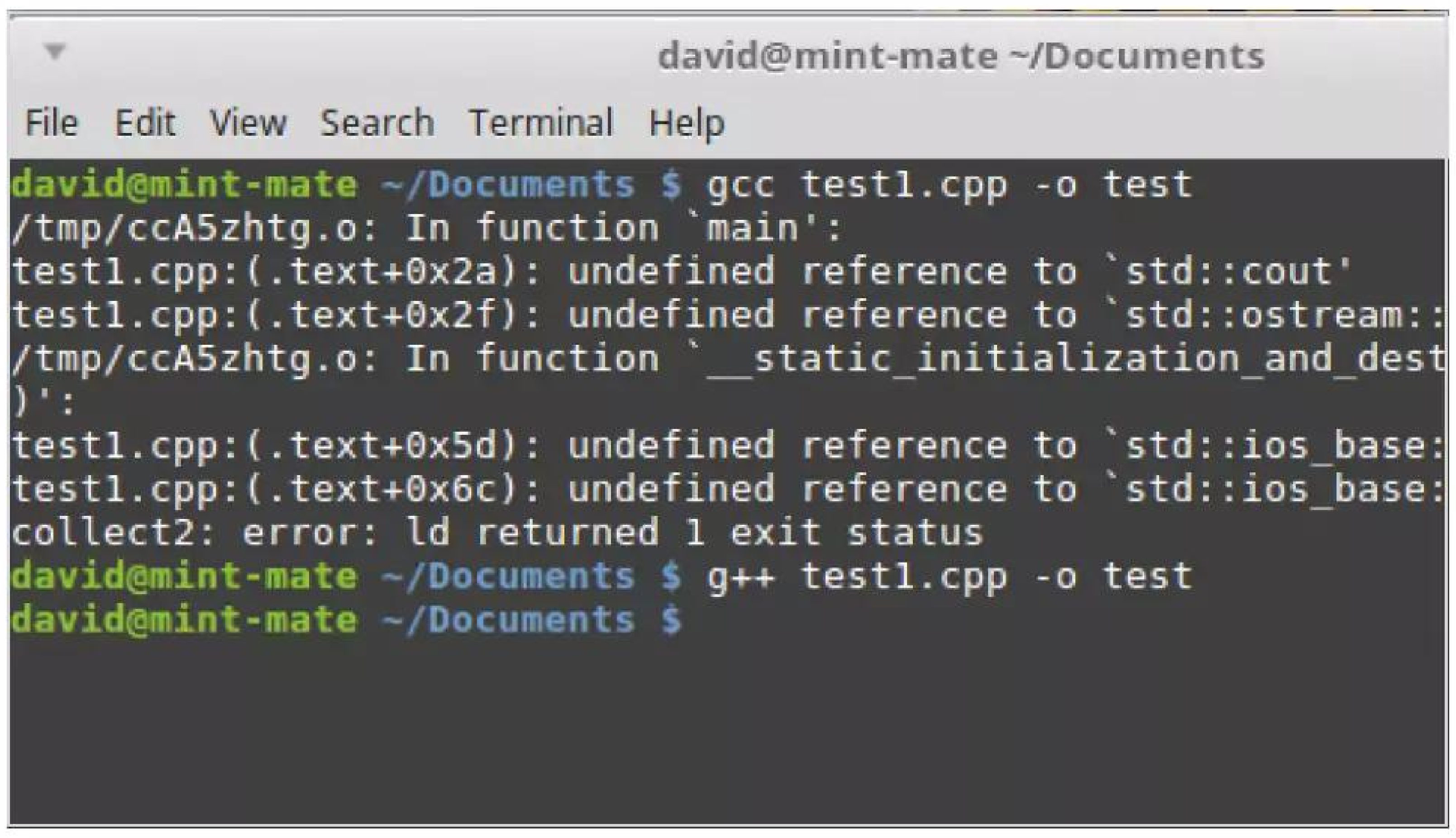
Under the #include part and simply using cout, cin and so on from then on.



```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    a=10;
    b=20;
    c=30;
    d=40;

    cout << a, b, c, d;
}
```

GCC OR G++

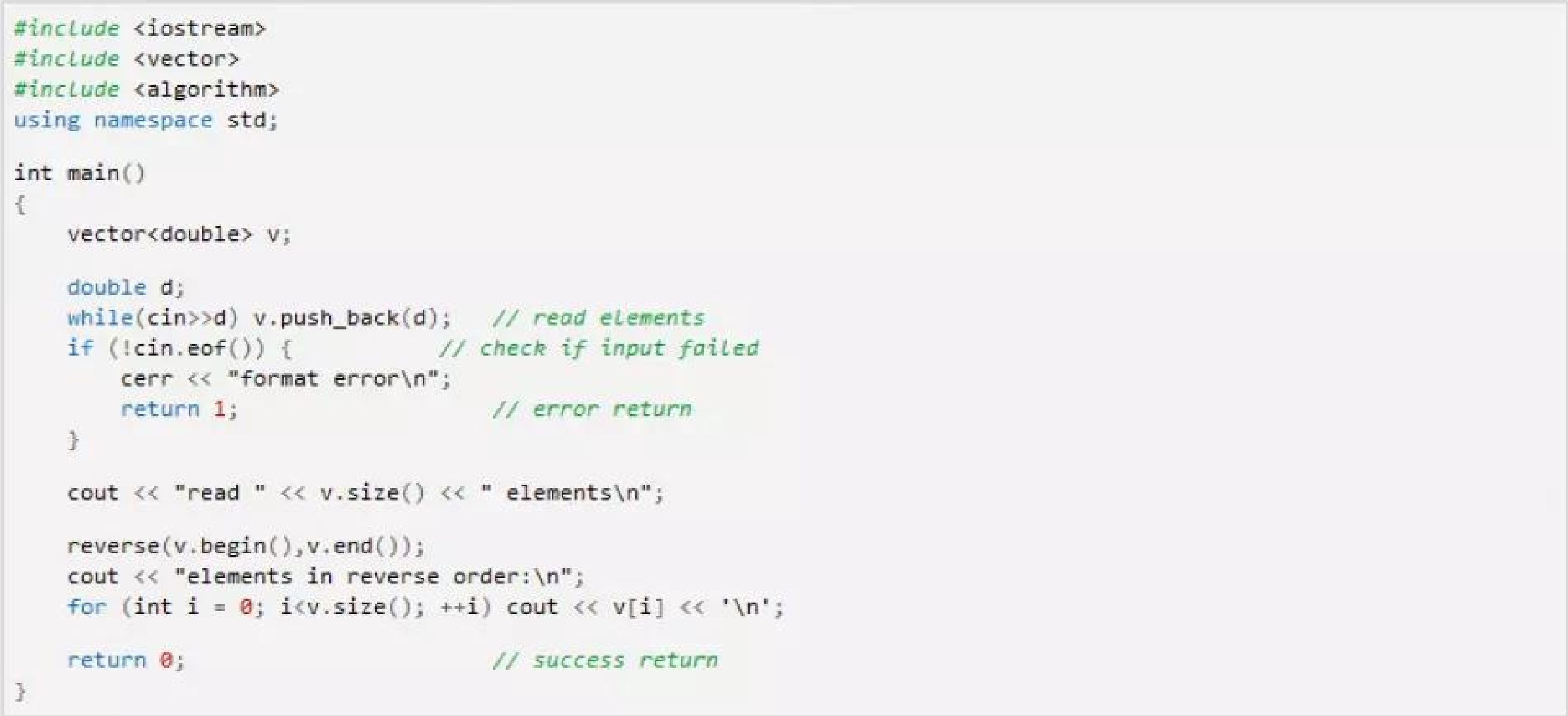
If you're compiling in Linux then you will no doubt come across gcc and g++. In short, gcc is the Gnu Compiler Collection (or Gnu C Compiler as it used to be called) and g++ is the Gnu ++ (the C++ version) of the compiler. If you're compiling C++ then you need to use g++, as the incorrect compiler drivers will be used.



```
david@mint-mate ~/Documents $ gcc test1.cpp -o test
/tmp/ccA5zhtg.o: In function `main':
test1.cpp:(.text+0x2a): undefined reference to `std::cout'
test1.cpp:(.text+0x2f): undefined reference to `std::ostream::
/tmp/ccA5zhtg.o: In function `__static_initialization_and_dest
)':
test1.cpp:(.text+0x5d): undefined reference to `std::ios_base:
test1.cpp:(.text+0x6c): undefined reference to `std::ios_base:
collect2: error: ld returned 1 exit status
david@mint-mate ~/Documents $ g++ test1.cpp -o test
david@mint-mate ~/Documents $
```

COMMENTS (AGAIN)

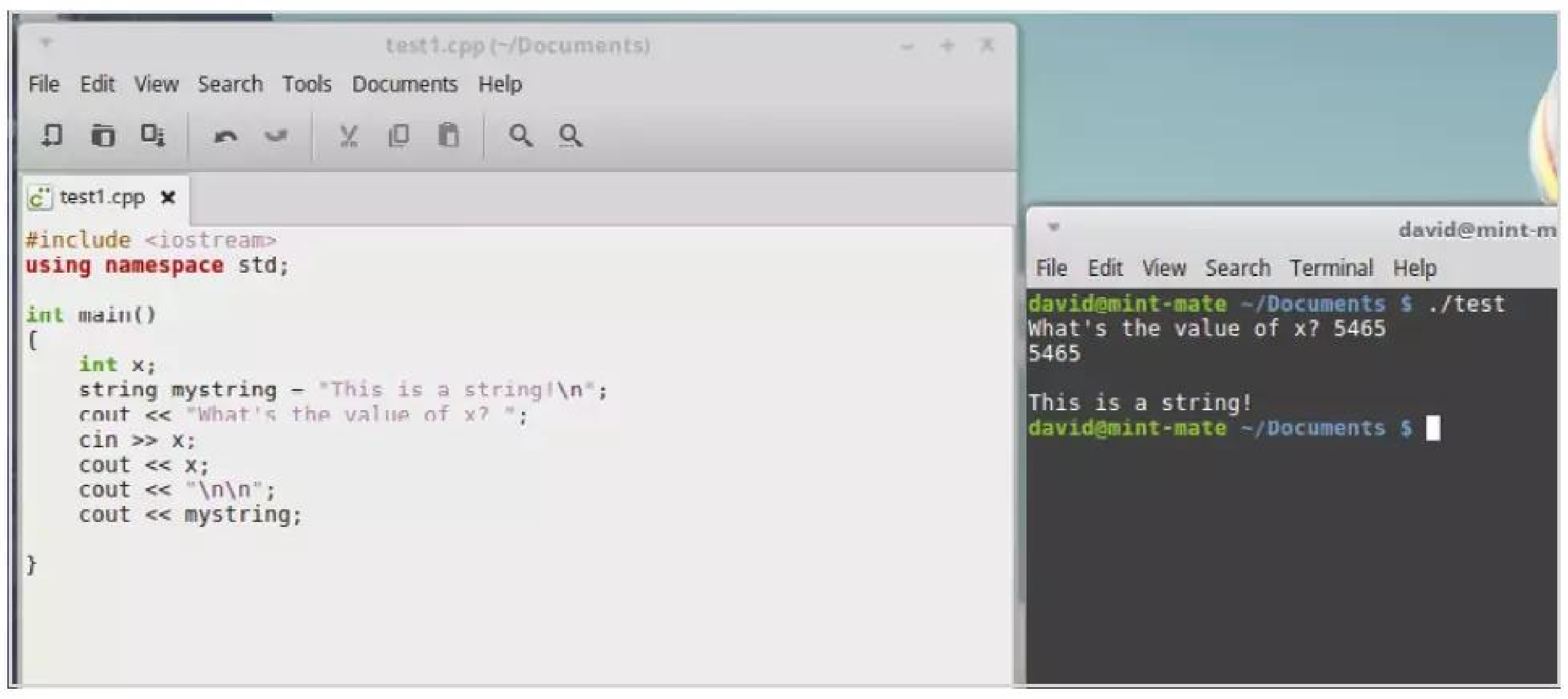
Indeed the mistake of never making any comments on code is back once more. As we've previously bemoaned, the lack of readable identifiers throughout the code makes it very difficult to look back at how it worked, for both you and someone else. Use more comments.



```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    vector<double> v;

    double d;
    while(cin>>d) v.push_back(d);   // read elements
    if (!cin.eof()) {           // check if input failed
        cerr << "format error\n";
        return 1;                   // error return
    }

    cout << "read " << v.size() << " elements\n";

    reverse(v.begin(),v.end());
    cout << "elements in reverse order:\n";
    for (int i = 0; i<v.size(); ++i) cout << v[i] << '\n';

    return 0;                       // success return
}
```

QUOTES

Missing quotes is a common mistake to make, for every level of user. Remember that quotes need to encase strings and anything that's going to be outputted to the screen or into a file, for example. Most compilers errors are due to missing quotes in the code.


test1.cpp (~/Documents)
File Edit View Search Tools Documents Help
test1.cpp
david@mint-m
File Edit View Search Terminal Help

```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    cout << "\n\n";
    cout << mystring;

}
```

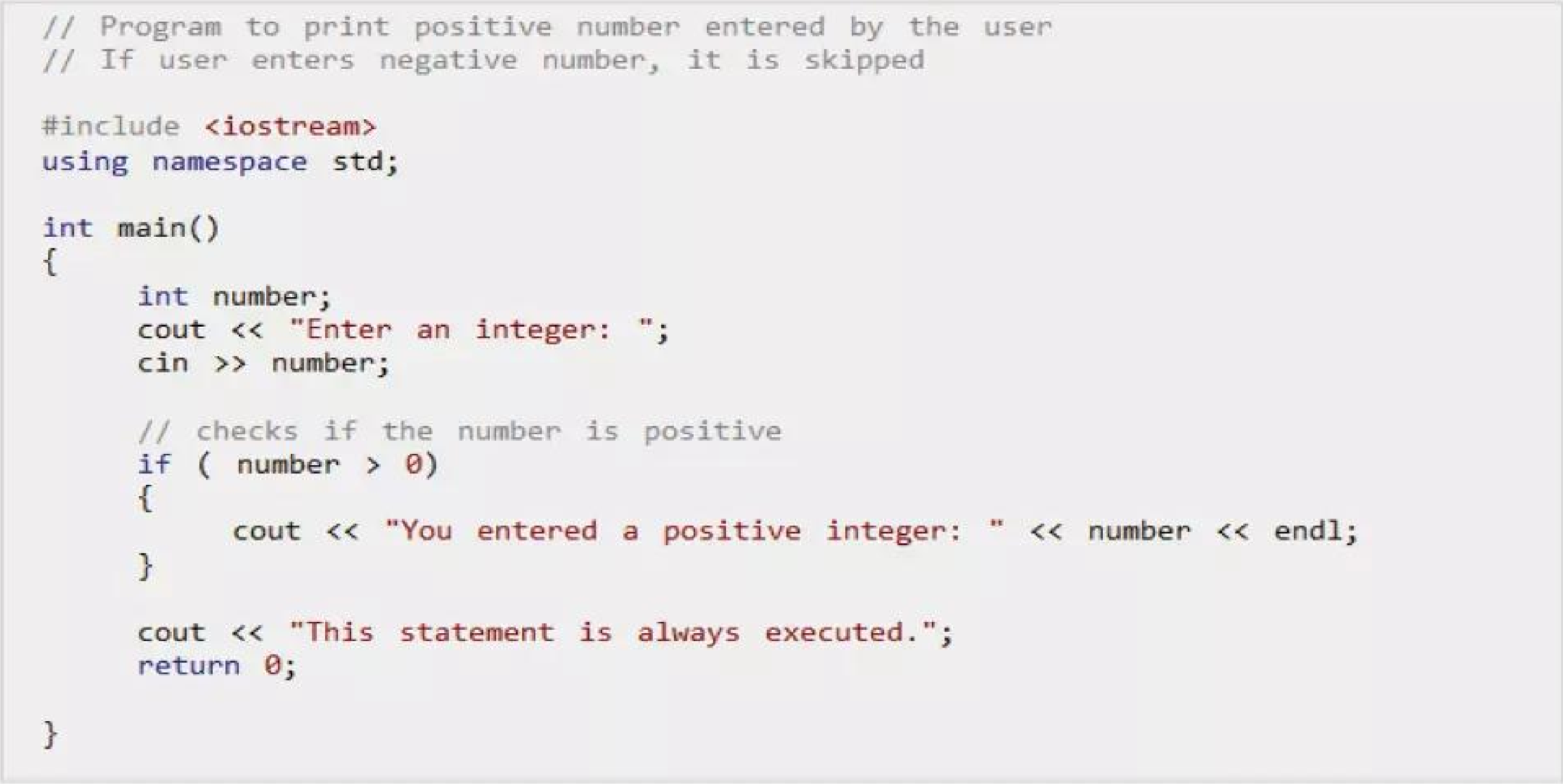
```
david@mint-mate ~/Documents $ ./test
What's the value of x? 5465
5465

This is a string!
david@mint-mate ~/Documents $
```

EXTRA SEMICOLONS

While it's necessary to have a semicolon at the end of every C++ line, there are some exceptions to the rule. Semicolons need to be at the end of every complete statement but some lines of code aren't complete statements. Such as:

```
#include
if lines
switch lines
```

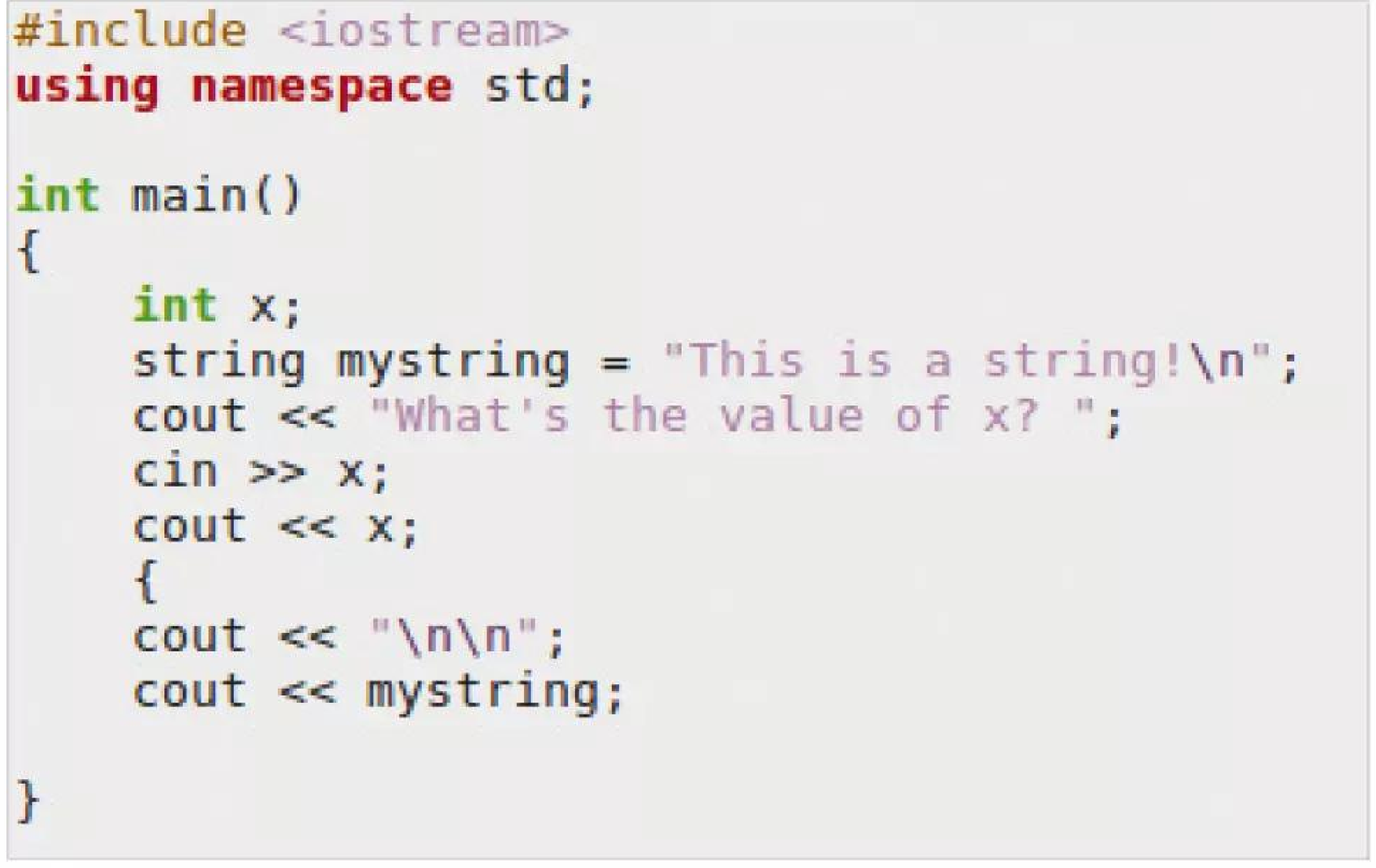
If it sounds confusing don't worry, the compiler lets you know where you went wrong.



```
// Program to print positive number entered by the user
// If user enters negative number, it is skipped

#include <iostream>
using namespace std;

int main()
{
     int number;
     cout << "Enter an integer: ";
     cin >> number;

     // checks if the number is positive
     if ( number > 0)
     {
          cout << "You entered a positive integer: " << number << endl;
     }

     cout << "This statement is always executed.";
     return 0;

}
```

TOO MANY BRACES

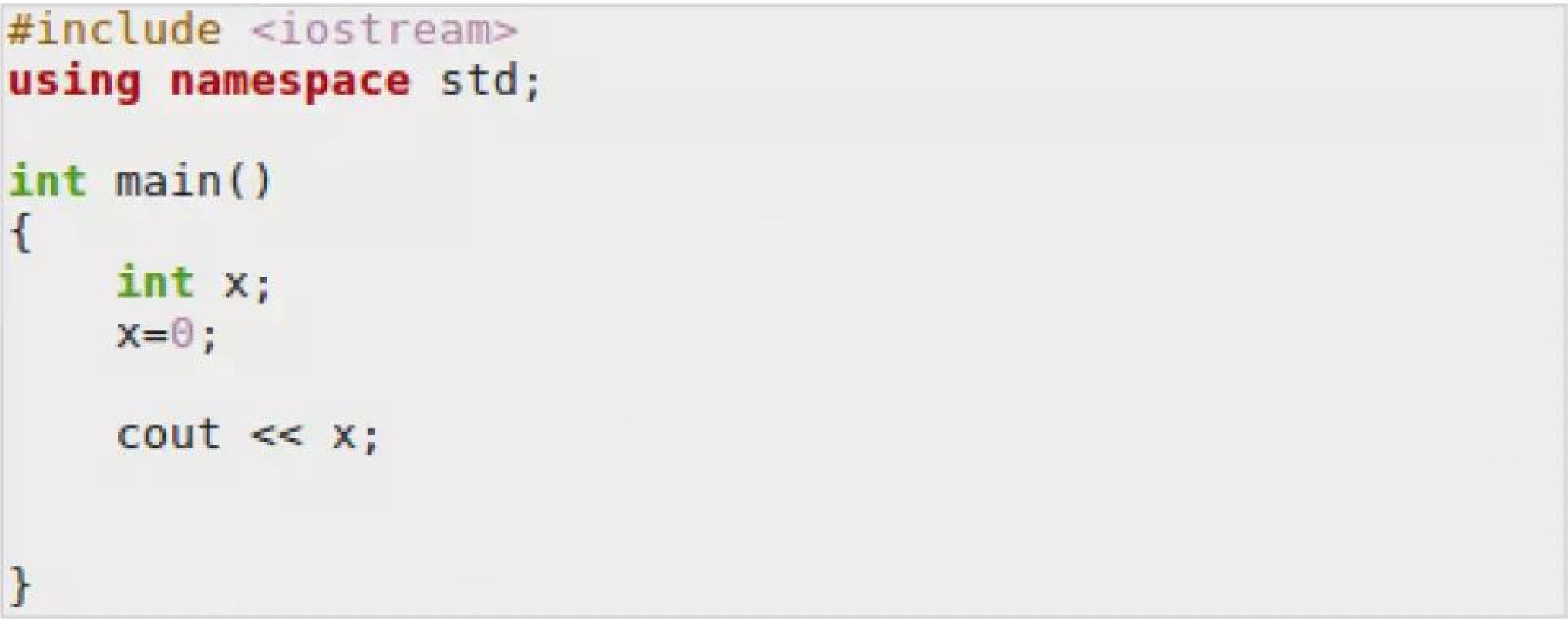
The braces, or curly brackets, are beginning and ending markers around blocks of code. So for every { you must have a }. Often it's easy to include or miss out one or the other facing brace when writing code; usually when writing in a text editor, as an IDE adds them for you.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    string mystring = "This is a string!\n";
    cout << "What's the value of x? ";
    cin >> x;
    cout << x;
    {
    cout << "\n\n";
    cout << mystring;

}
```

INITIALISE VARIABLES

In C++ variables aren't initialised to zero by default. This means if you create a variable called x then, potentially, it is given a random number from 0 to 18,446,744,073,709,551,616, which can be difficult to include in an equation. When creating a variable, give it the value of zero to begin with: `x=0`.



```
#include <iostream>
using namespace std;

int main()
{
    int x;
    x=0;

    cout << x;


}
```

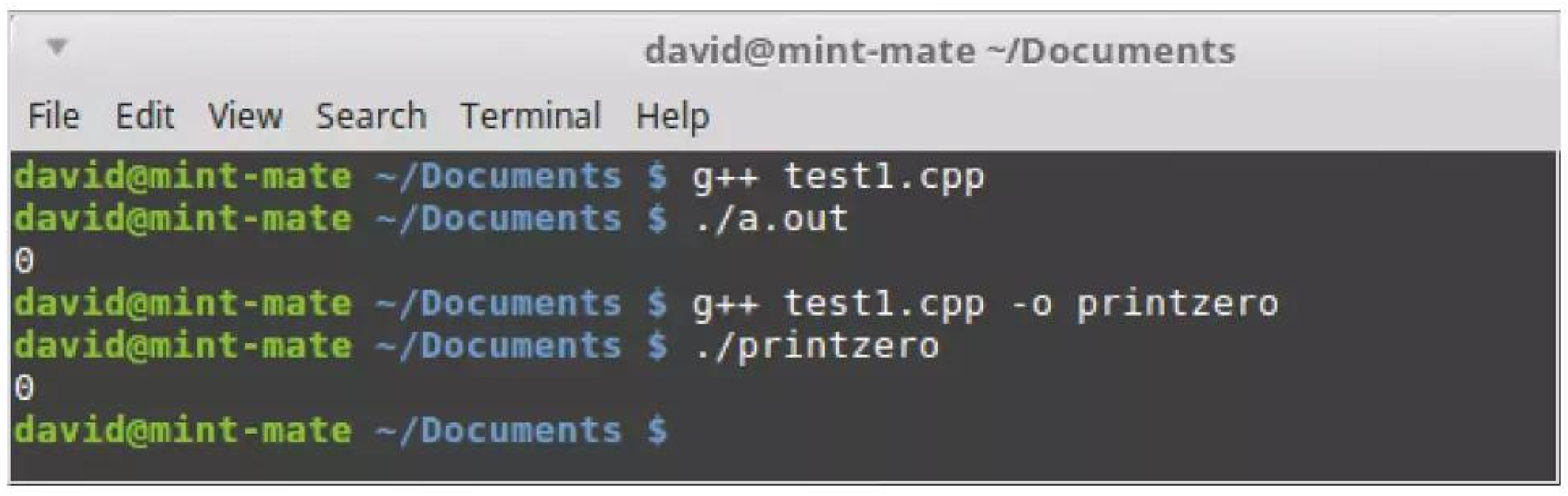
A.OUT

A common mistake when compiling in Linux is forgetting to name your C++ code post compiling. When you compile from the Terminal, you enter:

```
g++ code.cpp
```

This compiles the code in the file code.cpp and create an a.out file that can be executed with ./a.out. However, if you already have code in a.out then it's overwritten. Use:

```
g++ code.cpp -o nameofprogram
```


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ test1.cpp
david@mint-mate ~/Documents $ ./a.out
0
david@mint-mate ~/Documents $ g++ test1.cpp -o printzero
david@mint-mate ~/Documents $ ./printzero
0
david@mint-mate ~/Documents $
```

Where Next?

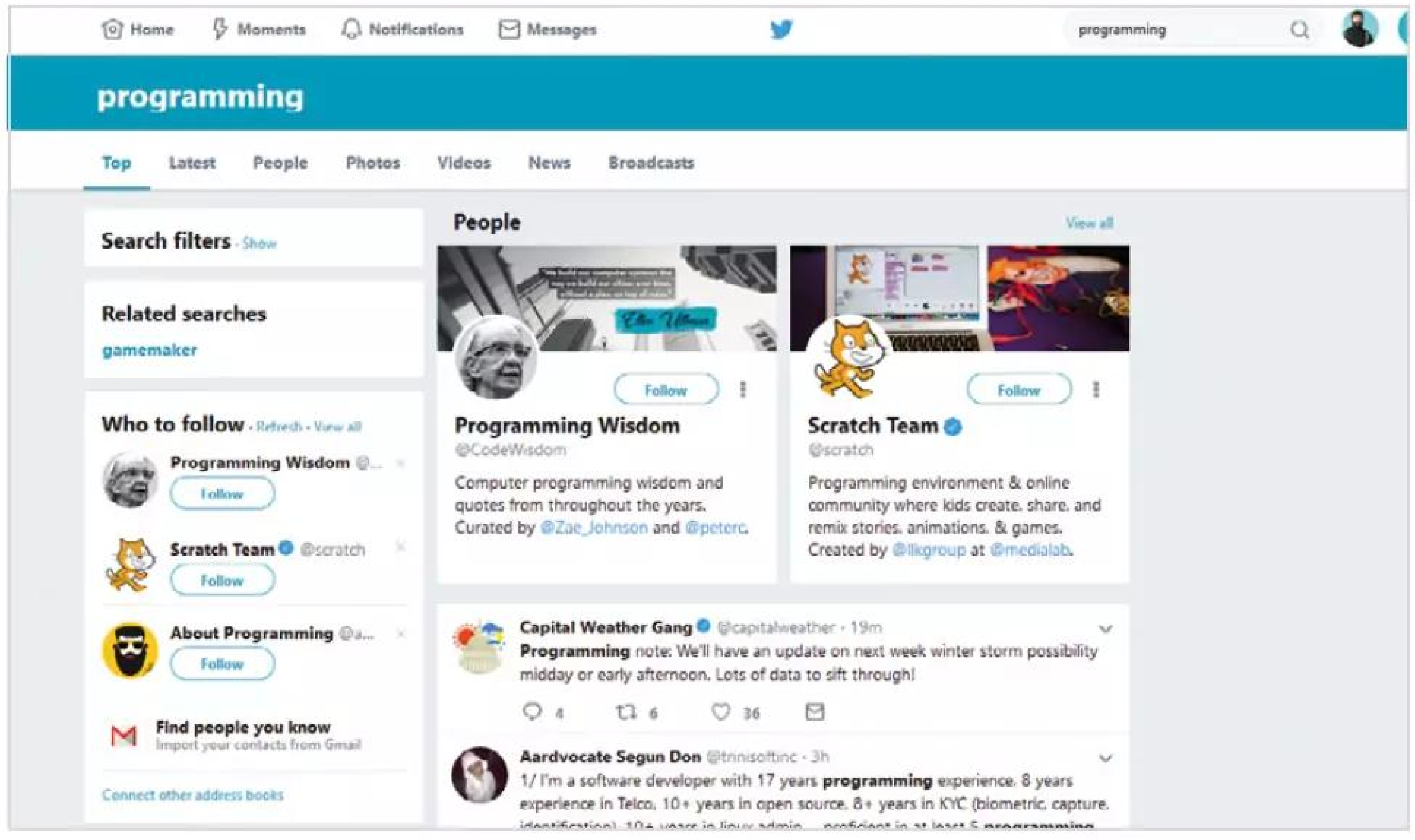
Coding, like most subjects, is a continual learning experience. You may not class yourself as a beginner any more but you still need to test your code, learn new tricks and hacks to make it more efficient and even branch out and learn another programming language.

#INCLUDE<KEEP ON LEARNING>

What can you do to further your skills, learn new coding practises, experiment and present your code and even begin to help others using what you've experienced so far?

TWITTER

Twitter isn't all trolls and antagonists, among the well-publicised vitriol are some genuine people who are more than willing to spread their coding knowledge. We recommend you find a few who you can relate to and follow them. Often they post great tips, hacks and fixes for common coding problems.


Home
Moments
Notifications
Messages
programming
programming
Top
Latest
People
Photos
Videos
News
Broadcasts
Search filters · Show
Related searches
gamemaker
Who to follow · Refresh · View all
Programming Wisdom @...
Follow
Scratch Team @scratch
Follow
About Programming @a...
Follow
Find people you know
Import your contacts from Gmail
Connect other address books
People
View all
Follow
Follow

Programming Wisdom
@CodeWisdom
Computer programming wisdom and quotes from throughout the years. Curated by @Zae_Johnson and @peterc.

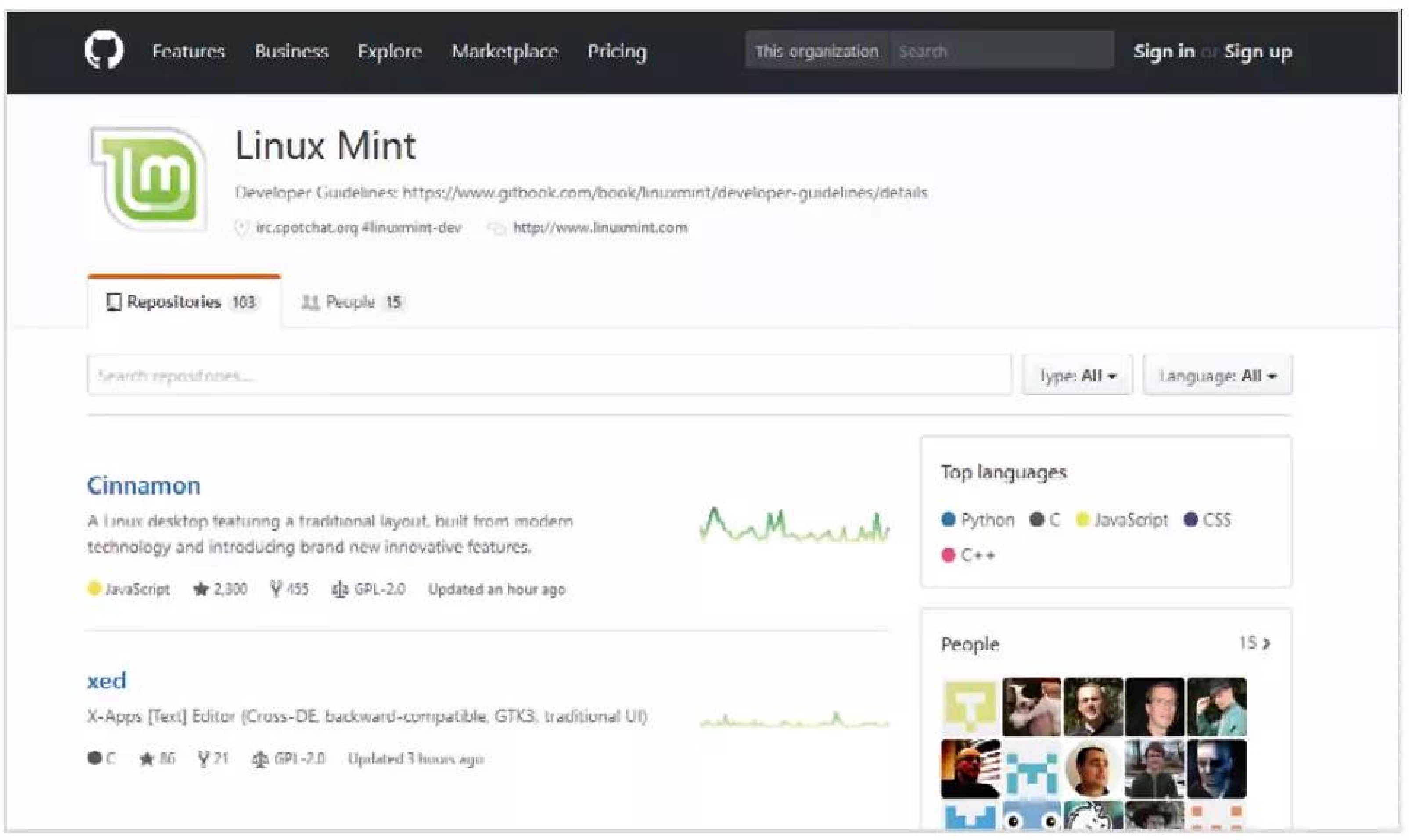
Scratch Team
@scratch
Programming environment & online community where kids create, share, and remix stories, animations, & games. Created by @llkgroup at @medialab.

Capital Weather Gang @capitalweather · 19m
Programming note: We'll have an update on next week winter storm possibility midday or early afternoon. Lots of data to sift through!
4 · 6 · 36

Aardvocate Segun Don @tnnisoftinc · 3h
1/ I'm a software developer with 17 years **programming** experience. 8 years experience in Telco, 10+ years in open source, 8+ years in KYC (biometric, capture,

OPEN PROJECTS

Look for open source projects that you like the sound of and offer to contribute to the code to keep it alive and up to date. There are millions of projects to choose from, so contact a few and see where they need help. It may only be a minor code update but it's a noble occupation for coders to get into.


Features
Business
Explore
Marketplace
Pricing
This organization
Search
Sign in or Sign up
Repositories 103
People 15
Search repositories...
Type: All
Language: All
Top languages
Python
C
JavaScript
CSS
C++
People
15

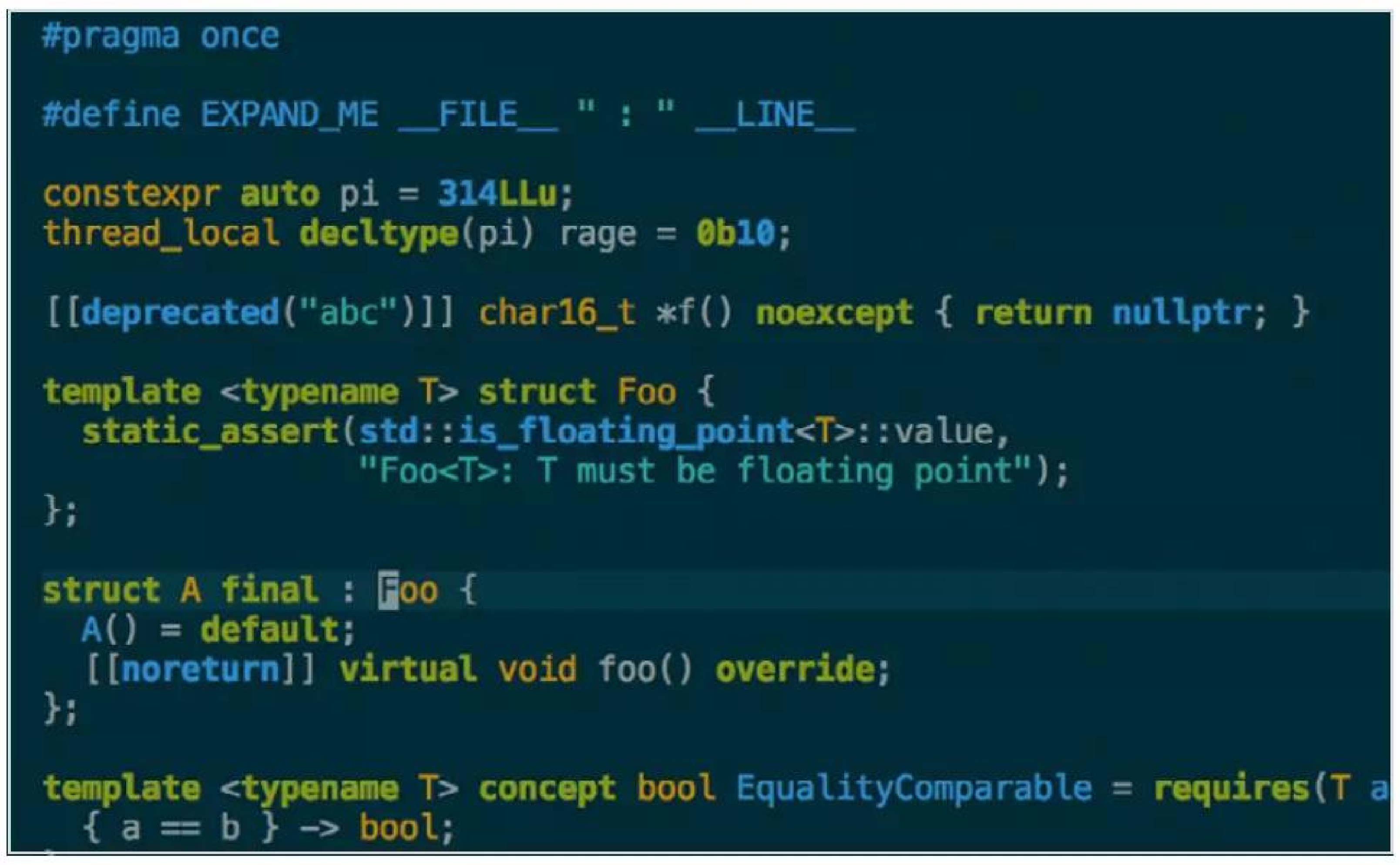
Linux Mint
Developer Guidelines: https://www.gitbook.com/book/linuxmint/developer-guidelines/details
irc.spotchat.org #linuxmint-dev · http://www.linuxmint.com

Cinnamon
A Linux desktop featuring a traditional layout, built from modern technology and introducing brand new innovative features.
JavaScript · 2,300 · 455 · GPL-2.0 · Updated an hour ago

xed
X-Apps [Text] Editor (Cross-DE, backward-compatible, GTK3, traditional UI)
C · 86 · 21 · GPL-2.0 · Updated 3 hours ago

KEEP CODING

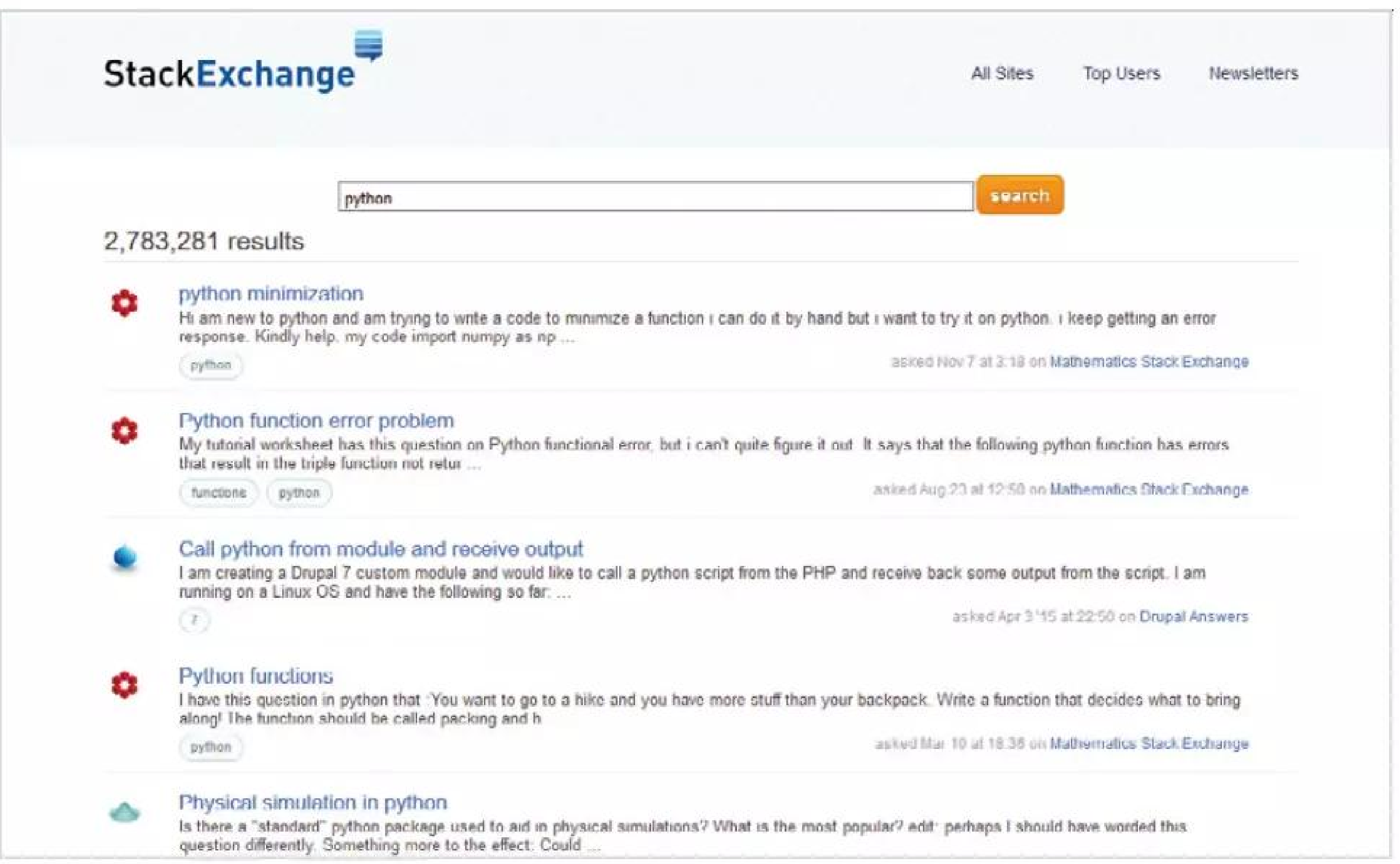
If you've mastered Python fairly well, then turn your attention to C++ or even C#. Still keep your Python skills going but learning a new coding language keeps the old brain ticking over nicely and give you a view into another community, and how they do things differently.



```
#pragma once

#define EXPAND_ME __FILE__ " : " __LINE__

constexpr auto pi = 314LLu;
thread_local decltype(pi) rage = 0b10;

[[deprecated("abc")]] char16_t *f() noexcept { return nullptr; }

template <typename T> struct Foo {
  static_assert(std::is_floating_point<T>::value,
                "Foo<T>: T must be floating point");
};

struct A final : Foo {
  A() = default;
  [[noreturn]] virtual void foo() override;
};

template <typename T> concept bool EqualityComparable = requires(T a
  { a == b } -> bool;
```

SHARE SKILLS

Become more active on coding and development knowledge sites, such as StackExchange. If you have the skills to start and help others out, not only will you feel really good for doing so but you can also learn a lot yourself by interacting with other members.


StackExchange
All Sites
Top Users
Newsletters
python
search

2,783,281 results

python minimization
Hi am new to python and am trying to write a code to minimize a function i can do it by hand but i want to try it on python. i keep getting an error response. Kindly help. my code import numpy as np ...
python
asked Nov 7 at 3:18 on Mathematics Stack Exchange

Python function error problem
My tutorial worksheet has this question on Python functional error, but i can't quite figure it out. It says that the following python function has errors that result in the triple function not retur ...
functions · python
asked Aug 23 at 12:50 on Mathematics Stack Exchange

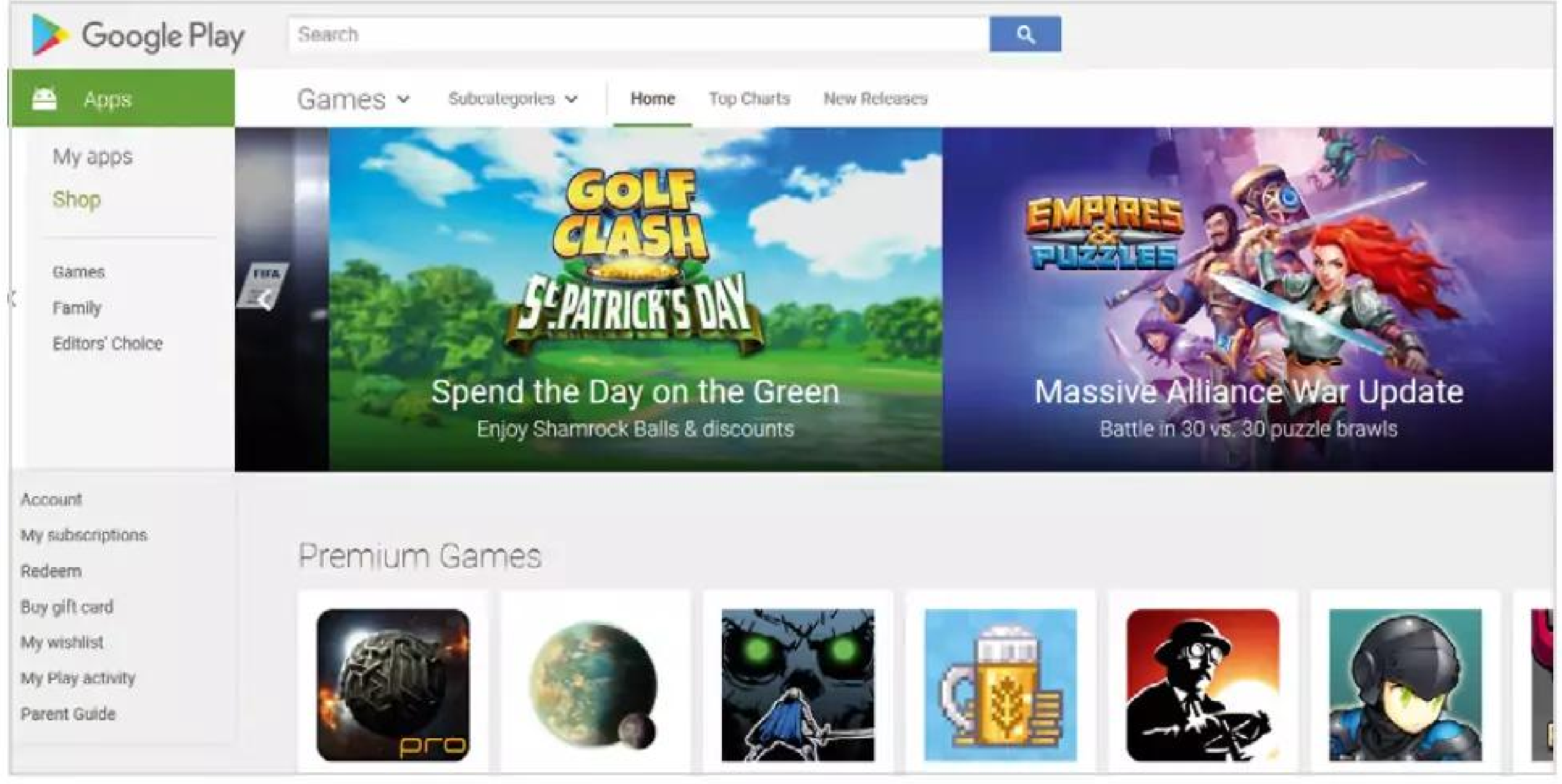
Call python from module and receive output
I am creating a Drupal 7 custom module and would like to call a python script from the PHP and receive back some output from the script. I am running on a Linux OS and have the following so far: ...
7
asked Apr 3 '15 at 22:50 on Drupal Answers

Python functions
I have this question in python that :You want to go to a hike and you have more stuff than your backpack. Write a function that decides what to bring along! The function should be called packing and h
python
asked Mar 10 at 18:38 on Mathematics Stack Exchange

Physical simulation in python
Is there a "standard" python package used to aid in physical simulations? What is the most popular? edit: perhaps I should have worded this question differently. Something more to the effect: Could ...

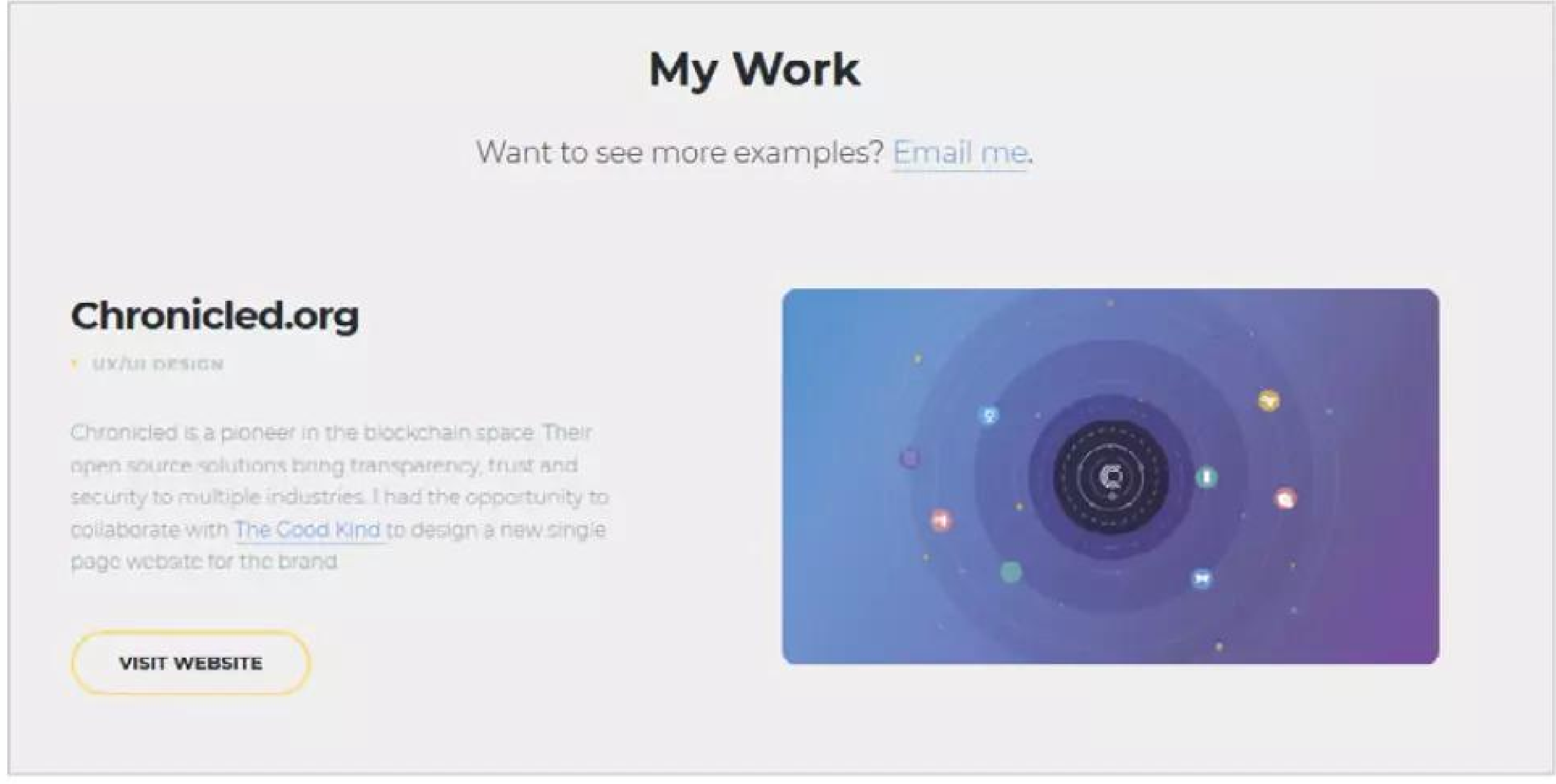
GOING MOBILE

The mobile market is a great place to test your coding skills and present any games or apps you've created. If your app is good, then who knows, it could be the next great thing to appear on the app stores. It's a good learning experience nevertheless, and something worth considering.


Google Play
Search
Apps
Games
Subcategories
Home
Top Charts
New Releases
My apps
Shop
Games
Family
Editors' Choice
GOLF CLASH
ST. PATRICK'S DAY
Spend the Day on the Green
Enjoy Shamrock Balls & discounts
EMPIRES & PUZZLES
Massive Alliance War Update
Battle in 30 vs. 30 puzzle brawls
Account
My subscriptions
Redeem
Buy gift card
My wishlist
My Play activity
Parent Guide
Premium Games

PORTFOLIOS

If you've learned how to code with an eye for a developer job in the future, then it's worth starting to build up an online portfolio of code. Look at job postings and see what skills they require, then learn and code something with those skills and add it to the portfolio. When it comes to applying, include a link to the portfolio.



My Work

Want to see more examples? Email me.

Chronicled.org

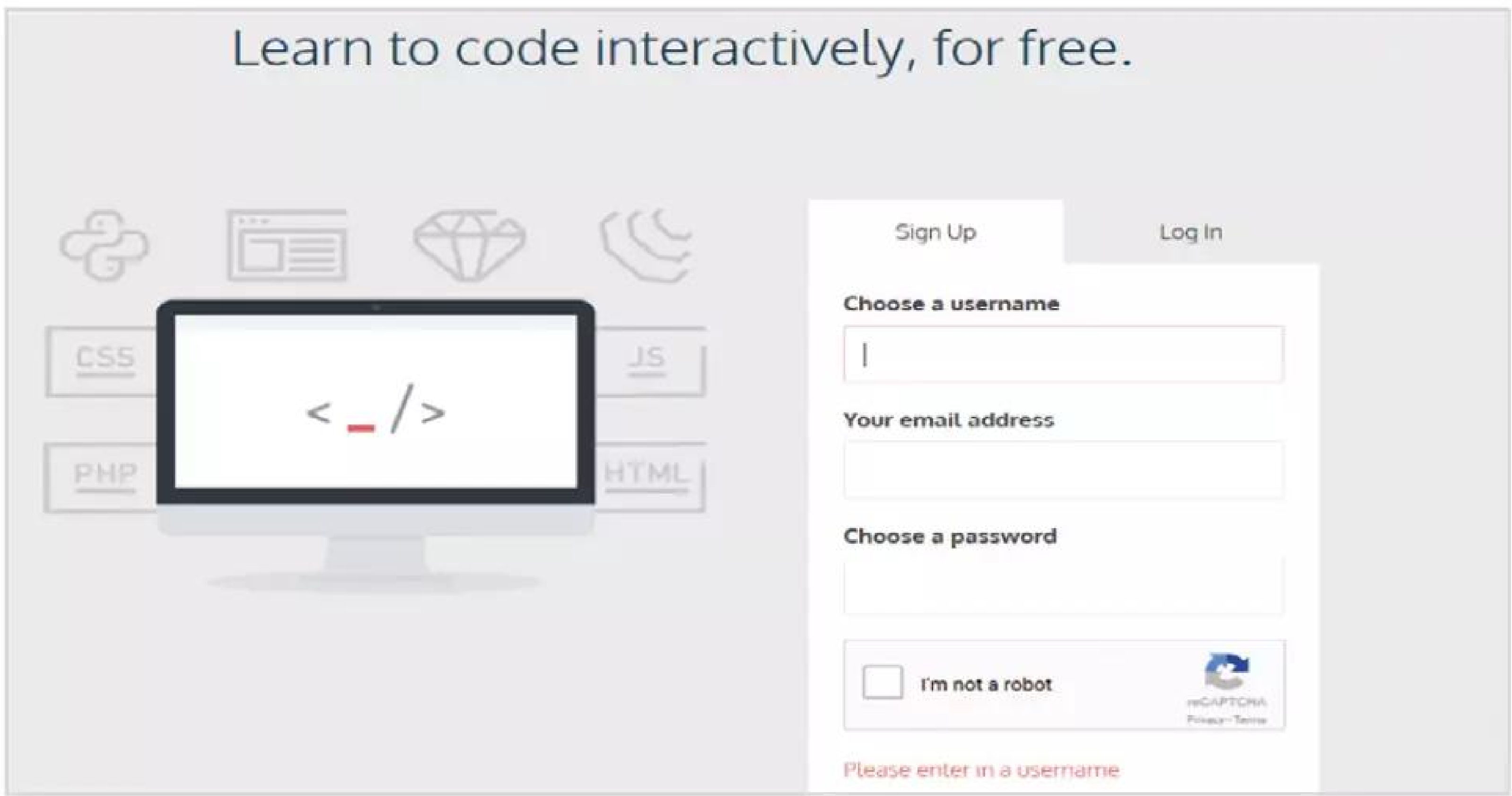
UX/UI DESIGN

Chronicled is a pioneer in the blockchain space. Their open source solutions bring transparency, trust and security to multiple industries. I had the opportunity to collaborate with The Good Kind to design a new single page website for the brand.

VISIT WEBSITE

ONLINE LEARNING

Online courses are good examples of where to take your coding skills next, even if you start from the beginner level again. Often, an online course follows a strict coding convention, so if you're self-taught then it might be worth seeing how other developers lay out their code, and what's considered acceptable.



Learn to code interactively, for free.

Choose a username

Your email address

Choose a password

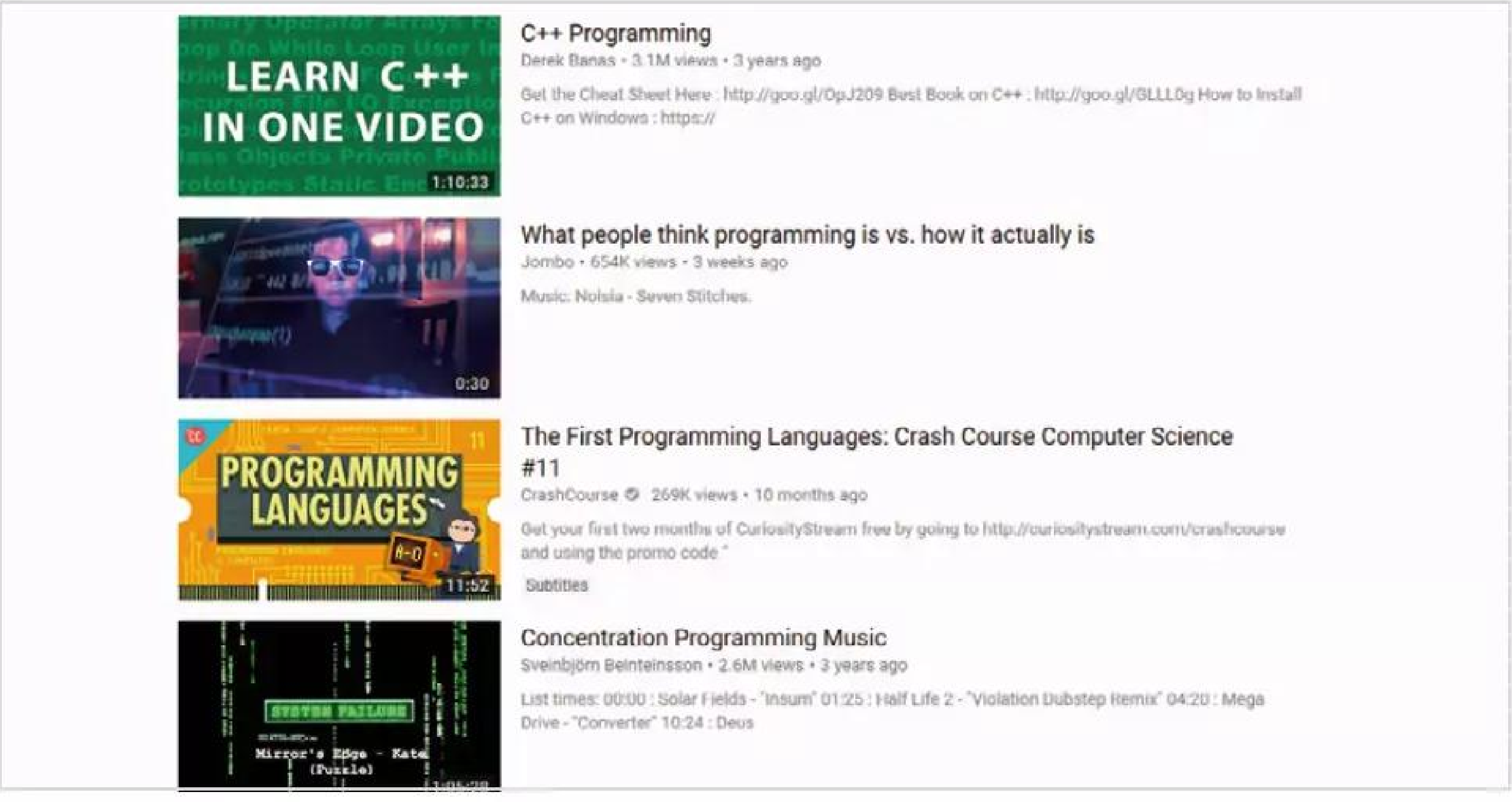
☐ I'm not a robot

Please enter in a username

CSS
JS
PHP
HTML
<_/>
Sign Up
Log In
reCAPTCHA

TEACH CODE

Can you teach? If your coding skills are spot on, consider approaching a college or university to see if they have need for a programming language teacher, perhaps a part-time or evening course. If not teaching, then consider creating your own YouTube how to code channel.



C++ Programming
Derek Banas • 3.1M views • 3 years ago
Get the Cheat Sheet Here : http://goo.gl/OpJ209 Best Book on C++ : http://goo.gl/GLLL0g How to Install C++ on Windows : https://

What people think programming is vs. how it actually is
Jombo • 654K views • 3 weeks ago
Music: Noisia - Seven Stitches.

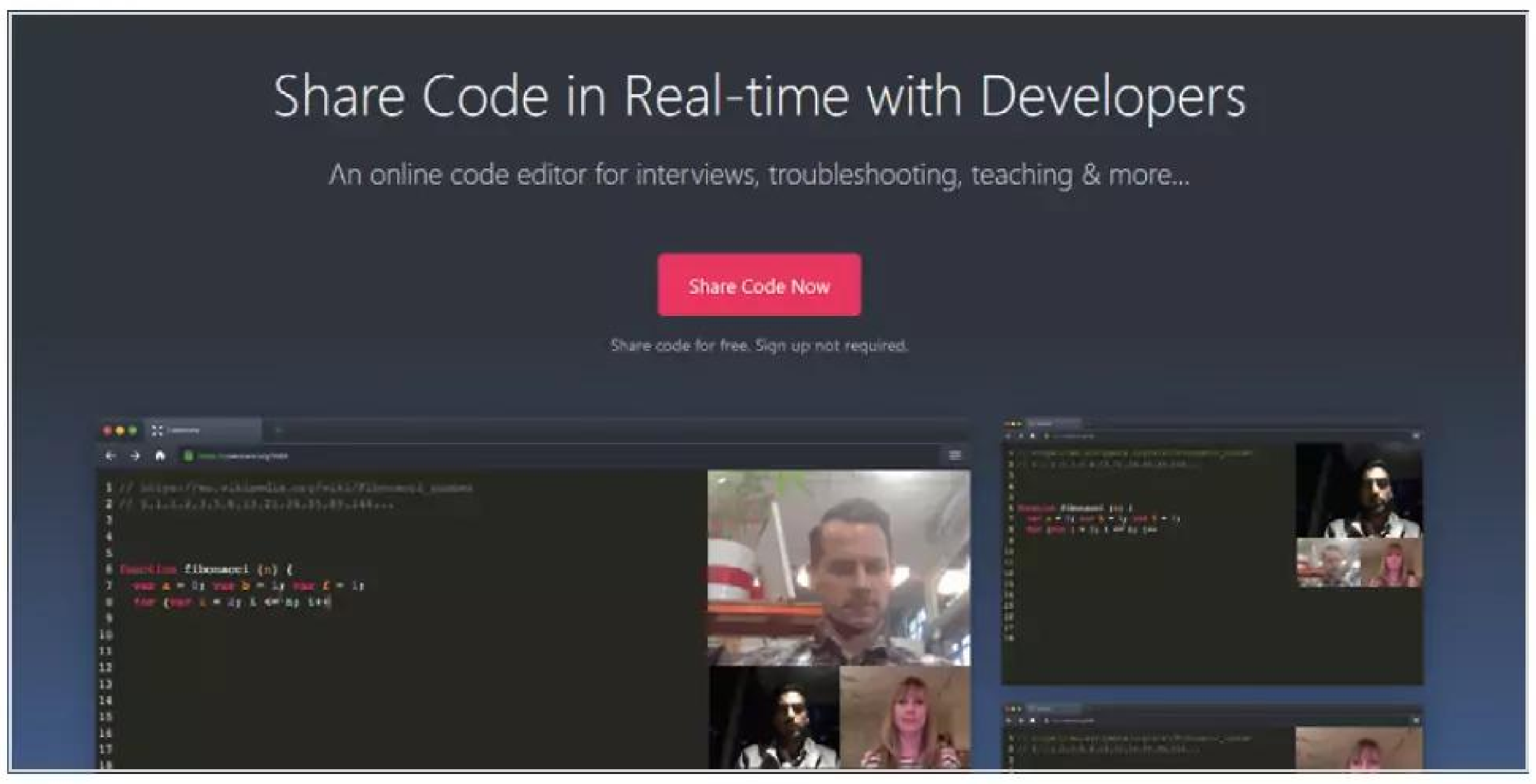
The First Programming Languages: Crash Course Computer Science #11
CrashCourse • 269K views • 10 months ago
Get your first two months of CuriosityStream free by going to http://curiositystream.com/crashcourse and using the promo code "
Subtitles

Concentration Programming Music
Sveinbjörn Beinteinsson • 2.6M views • 3 years ago
List times: 00:00 : Solar Fields - "Insum" 01:25 : Half Life 2 - "Violation Dubstep Remix" 04:20 : Mega Drive - "Converter" 10:24 : Deus

LEARN C++ IN ONE VIDEO
1:10:33
0:30
PROGRAMMING LANGUAGES
11:52

SHARE CODE

Get sharing, even if you think your code isn't very good. The criticism, advice and comments you receive back help you iron out any issues with your code, and you add them all to your checklist. Alternatively your code might be utterly amazing but you won't know unless you share it.



Share Code in Real-time with Developers

An online code editor for interviews, troubleshooting, teaching & more...

Share code for free. Sign up not required.

Share Code Now

HARDWARE PROJECTS

Contributing to hardware projects is a great resource for proving your code with others and learning from other contributors. Many of the developer boards have postings for coders to apply to for hardware projects, using unique code to get the most from the hardware that's being designed.

Sat, Oct 28 04:12 20 pm
13°F
Windchill: 13°F
Windspeed: calm mph
Direction: CALM
Barometer: 1012.87"Hg
Humidity: 69%
Today:
Sunday:
Monday:
Tuesday:
20°F/17°F 0%
19°F/16°F 100%
WINDY 13°F/8°F 20%
14°F/6°F 0%

Want to master your Code?
Then don't miss our NEW Programming magazine on Readly now!

NEW
Bumper Holiday Special! BONUS DOUBLE EDITION!
The Future Is Code
Coding
USER
Issue Eight | December 2023 | 100% INDEPENDENT
Learn To Code Now!
The Essential User Guide for the First Time Programmer
FREE SPECIAL BONUS
TECH SHOPPER MAGAZINE
FULL 170 Page Magazine!
Give Your Career A Mega Boost!
Which Coding Platform is Right for You?
Start Your First Coding Project Inside!
Discover New Skills to Improve Your Work Life.
Our Experts Help You Make the Right Choice.
Avoid any Errors & get the Right Start with Coding.

Click our handy link to read now: https://bit.ly/3OcL1zx

C++ & Python Tricks & Tips
21 | ISBN: 978-1-912847-54-9

Published by: Papercut Limited
Digital distribution by: Readly AB

©2025 Papercut Ltd. All rights reserved. No part of this publication may be reproduced in any form, stored in a retrieval system or integrated into any other publication, database or commercial programs without the express written permission of the publisher. Under no circumstances should this publication and its contents be resold, loaned out or used in any form by way of trade without the publisher's written permission. While we pride ourselves on the quality of the information we provide, Papercut Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. Due to the nature of the tech industry, the publisher cannot guarantee that all apps and software will work on every version of device. It remains the purchaser's sole responsibility to determine the suitability of this book and its contents for whatever purpose. Any app, hardware or software images reproduced on the front cover are solely for design purposes and are not necessarily representative of content. We advise all potential buyers to check listings prior to purchase for confirmation of actual content.
All editorial herein is that of the reviewer - as an individual - and is not representative of the publisher or any of its affiliates. Therefore the publisher holds no responsibility in regard to editorial opinion or content.

This is an independent publication and as such does not necessarily reflect the views or opinions of the producers of apps, software or products contained within. This publication is 100% unofficial and in no way associated with any other company, app developer, software developer or manufacturer. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged. Relevant graphic imagery reproduced with courtesy of brands, apps, software and product manufacturers. Additional images are reproduced under licence from Shutterstock. Prices, international availability, ratings, titles and content are subject to change. Some content may have been previously published in other editions. All information was correct at time of publication.

Papercut Limited
Registered in England & Wales No: 04308513

ADVERTISING - For our latest media packs please contact:
Brad Francis - brad@papercutltd.co.uk
Web - www.pclpublications.com

INTERNATIONAL LICENSING - Papercut Limited has many great publications and all are available for licensing worldwide.
For more information email: jgale@pclpublications.com